University of Zurich[UZH]

Department of Informatics

Supporting Requirements and Acceptance Tests Alignment During Software Evolution

Dissertation submitted to the Faculty of Business,
Economics and Informatics of
the University of Zurich

to obtain the degree of
Doktorin der Wissenschaften, Dr. sc.
(corresponds to Doctor of Science, PhD)

presented by Sofija Hotomski
from Serbia

approved in April 2019

at the request of
Prof. Martin Glinz, Dr. rer. nat.
Prof. Fabiano Dalpiaz, Ph.D.

University of Zurich[UZH]

The Faculty of Business, Economics and Informatics of the University of Zurich hereby authorizes the printing of this dissertation, without indicating an opinion of the views expressed in the work.

Zurich, April 03, 2019

Chairman of the Doctoral Board: Prof. Dr. Thomas Fritz

The good life is one inspired by love
and guided by knowledge.

—Bertrand Russell

(Philosopher, 1872 - 1970)

Science is organized knowledge.
Wisdom is organized life.

—Emanuel Kant

(Philosopher, 1724 - 1804)

Acknowledgement

Doing a PhD is equivalent to commitment, persistence and hard work. All of these are essential, but insufficient for one to successfully finish a PhD journey. Without support of people around me I would never have been able to come this far.

First and foremost I would like to thank Prof. Dr. Martin Glinz for providing me with the opportunity to do my thesis in his research group. I am grateful for his continuous support, valuable feedback and interesting discussions we had over the past years. He supported my ideas and guided me through the paths even when they became uncertain. I am thankful for the freedom he gave me to explore new ideas and different research areas. Finally, his kindness and empathy gave me the feeling that I belong to this group.

I express my sincere gratitude to Prof. Dr. Fabiano Dalpiaz for accepting to co-advise my thesis.

I would also like to acknowledge the Swiss National Science Foundation (SNSF) for partially supporting my PhD work (grant 200021-157004/1). Further I thank all companies which supported the GuideGen project and gave valuable feedback throughout these years.

I want to thank my RERG colleagues for exchanging experiences and thoughts about (PhD) life and many other topics. I am truly grateful to Eya Ben Charrada for helping me with writing and refining my papers and giving me continued encouragements. I am thankful to Dustin Wüest, who reviewed my papers and made the working place much more fun together with Norbert Seyff, Irina Todoran Koitz, Martina Kolpondinos-Huber, Parisa Ghazi, Emitzá Guzmán and Melanie Stade. I would also like to thank to my colleagues from the Department of Informatics for the nice and mindful discussions. I am thankful to Prof. Dr. Renato Pajarola and his research group for many interesting lunch talks.

I would like to thank my dad, whose work inspired me on this journey. I am grateful to my mum for her endless support and encouragements and to my sisters Olga, Mirjana and Tamara for reviewing my work and teaching me a lot about work and life.

I would like to thank my husband Slobodan for his unconditional love and support, for providing feedback on my research ideas and for his constant encouragements.

I am especially grateful to my friend Boris Spasojević for notifying me about this PhD position and supporting me all way through this journey. I want to thank Dunja Topalov and Aleksandar Risović who helped with designing my application. I am grateful to them and to my friends Ivan Dobrić, Vladan Marsenić, Nikola Živković, Nemanja Maločić, Iva Popov, Jelena Polinger, Zorana Milošević, Dragana Čalija, Milica Nenadić, Nemanja Škorić, Ivana Matić and others who believed in me even when I did not.

Thank you all for being there for me!

Abstract

When software-based systems evolve, their requirements constantly change. Changes in requirements frequently affect their related acceptance tests. However, even when modifications are applied to requirements documents, they are often not accordingly applied to the affected acceptance tests, leaving them outdated and non-aligned with the changed requirements. Keeping requirements and acceptance test documents up-to-date and aligned with each other is challenging due to poor documentation, communication practices, and insufficient tool support. Having such outdated and inconsistent documentation may introduce software quality problems, unintended costs and project delays. In this thesis, we contribute to improving the current state of practice with regard to requirements and acceptance tests management in industry with a novel approach for keeping requirements aligned with their acceptance tests via automatically generated guidance. Our tool-supported approach combines information retrieval (IR) and natural language processing (NLP) concepts to provide suggestions about how to adapt the affected acceptance tests when their requirements change. With such suggestions, our

approach facilitates the alignment of acceptance tests with the actual requirements. In addition, our approach provides e-mails and warnings, which remind requirements engineers and testers about mismatches in the documents, thus reducing the risk of non- and miscommunicated changes.

This thesis contributes the insights into requirements and acceptance tests management in industry, the conceptual solution of our approach, the GuideGen tool as a proof of concept and two evaluations of the approach - quantitative and qualitative, respectively. All studies are conducted by using real-world data and with practitioners from industry. The results from the first, exploratory study point out that acceptance tests are not regularly maintained, which occasionally results in confusing features and bugs. The second study shows that our approach provides useful guidance for maintaining acceptance tests and keeping them aligned with their evolving requirements. The notification system facilitates the communication of changes among different stakeholders and serves as a good reminder for required adaptations. In the two evaluation studies, several participants pointed out that GuideGen is a practically useful tool that facilitates maintaining the requirements and acceptance tests alignment and communicating changes when requirements evolve.

Zusammenfassung

Wenn softwarebasierte Systeme sich weiterentwickeln, ändern sich deren Anforderungen ständig. Anforderungsänderungen wirken sich häufig auf die damit verbundenen Abnahmetests aus. Bei der Änderung von Anforderungen werden jedoch die betroffenen Abnahmetests häufig nicht aktualisiert, so dass diese veralten und nicht mehr mit den geänderten Anforderungen konsistent sind. Die Aktualisierung und Abstimmung von Anforderungs- und Abnahmetestunterlagen ist aufgrund schlechter Dokumentations- und Kommunikationspraktiken und unzureichender Werkzeugunterstützung eine Herausforderung. Eine solche veraltete und inkonsistente Dokumentation kann zu Qualitätsproblemen bei der Software, unbeabsichtigten Kosten und Projektverzögerungen führen. In dieser Arbeit tragen wir dazu bei, den aktuellen Stand der Praxis in Bezug auf das Management von Anforderungen und Abnahmetests in der Industrie zu verbessern, indem wir einen neuartigen Ansatz verfolgen, um die Anforderungen mit ihren Abnahmetests über automatisch generierte Änderungsempfehlungen in Einklang zu bringen. Unser werkzeuggestützter Ansatz kombiniert Konzepte des Information Retrieval und der Verarbeitung

natürlicher Sprache und generiert Empfehlungen, wie die betroffenen Abnahmetests angepasst werden können, wenn sich ihre zugehörigen Anforderungen ändern. Mit solchen Empfehlungen erleichtert unser Ansatz die Anpassung der Abnahmetests an die geänderten Anforderungen. Zusätzlich bietet unser Ansatz E-Mails und Warnungen, die die Anforderungs- und Testingenieure an Abweichungen in den Unterlagen erinnern und so das Risiko von nicht oder falsch kommunizierbaren Änderungen reduzieren.

Diese Dissertation liefert mehrere wissenschaftliche Beiträge: erstens, Erkenntnisse über das Management von Anforderungen und Abnahmetests in der Industrie; zweitens, die prototypische Implementierung des GuideGen Werkzeugs; und drittens, zwei empirische Evaluationen – eine quantitative und eine qualitative. Alle Studien verwenden reale Daten aus der Industrie. Die Ergebnisse der ersten Forschungsstudie zeigen, dass Abnahmetests nicht regelmäßig durchgeführt werden, was gelegentlich zu verwirrenden Merkmalen und Fehlern führt. Die zweite Studie zeigt, dass unser Ansatz eine nützliche Hilfestellung für die Pflege von Abnahmetests und deren Konsistenthaltung mit veränderten Anforderungen bietet. Das Benachrichtigungssystem erleichtert die Kommunikation der Änderungen zwischen verschiedenen Interessengruppen und dient als gute Erinnerung an notwendige Anpassungen. In den beiden Evaluationsstudien haben eine Reihe von Teilnehmern betont, dass GuideGen ein praktisch nützliches Werkzeug ist, das die Erhaltung der Konsistenz zwischen Anforderungen und Abnahmetests sowie die Kommunikation von Anforderungsänderungen erleichtert.

Contents

List of Tables

List of Figures

Chapter 1

Synopsis

Keeping software requirements and acceptance test documents up-to-date and consistent when software systems evolve plays a key role in software maintainability. The purpose of requirements documents is to provide a means of communication among different stakeholders, supporting system-testing activities and controlling the evolution of the system [Dav93]. There are different types of requirements documents. According to Wiegers [WB13], business requirements are kept in a vision and scope document, user requirements are kept in a use case document, while functional and non-functional requirements reside in a software requirements specification (SRS). In Agile projects, business requirements are documented in epics, user and functional requirements are documented as features and user stories, while non-functional requirements are defined as backlog constraints [Lef10]. In this work we focus on documents that contain user, functional and non-functional requirements rather than high-level business goals.

Acceptance test documents are used for comparing a system to its initial requirements and the current needs of its end users [MSB11] and for verifying whether the system implements the requirements as intended [Lef10]. There are two main types of acceptance test documents: manual and automated (executable). They are both written in natural language, but using different levels of formalism. Manual acceptance tests contain steps and expected results that are written without following any formal rules. In contrast, automated acceptance tests follow a formal syntax, such as Gherkin [WHT17], and are written in the *Given-When-Then* form. This work concentrates on manual, non-executable acceptance test artifacts, since they are used in industry more frequent than the automated ones.

Requirements and acceptance test artifacts show much synergy, as acceptance tests are written based on requirements documents [HBCG16], [Lef10] and they both refer to system's behavior. Therefore, changes in requirements documents usually affect their acceptance tests and the tests should be adapted accordingly. In practice, however, the affected acceptance tests are not always modified, especially in smaller, centralized teams with good communication [HBCG16].

Many factors lead to inconsistencies between requirements and acceptance tests documents. For instance, when system requirements change, the management of acceptance tests is hindered by poor communication of changes between requirements and test engineers and low quality of requirements documents [HG18b], [Adz09]. Next, testers are usually excluded from negotiations about requirements [HBCG16], [BRB+14]. When requirements engineers/product owners are not immediately available, due to time pressure test engineers

often make their own assumptions about the changes in requirements, which can lead to inadequate adaptations of acceptance tests [HG18b]. Insufficient tool support also hinders the documentation update. For instance, currently used tools do not offer a centralized solution for managing both, requirements and acceptance test documents synchronously. Moreover, they do not provide features for automated traceability, change impact analysis or change propagation. Making such traces and identifying which documents are affected by changes in requirements and how they should be adapted still requires manual effort. Moreover, in order to correctly adapt affected acceptance tests, test engineers require further communication with requirements engineers, which can be error-prone and time consuming.

In this thesis we propose a solution which provides concrete suggestions about how the affected acceptance tests should be adapted when their requirements change. Although the alignment of requirements and acceptance tests depends to a large degree on the organizational culture and motivation for applying alignment practices [BRB+14], [HBCG16], we think that having a tool support and concrete guidance can facilitate this process and motivate practitioners to keep their documents up to date and consistent.

1.1 State of the Art

In this section, we give an overview of the state of the art regarding approaches for keeping requirements and other software artifacts up-to-date, traceable and aligned. In particular, we show that there is a need for innovative solutions with regard to managing changes in software requirements and propagating changes to other software artifacts. Although the existing approaches are, to a large extent, able to automatically provide correct traces among documents and identify artifacts affected by a change in a requirement, none of them provide concrete guidance about how these documents should be adapted.

1.1.1 Requirements change management

Heraclitus' assertion saying that *change is the only constant* is applicable to software engineering activities. In particular, changes in requirements are inevitable and, if not managed correctly, they may lead to immoderate costs, project delays, or even project failures [BMS02], [CKI88]. According to Stark et al. [SOSA99], the basic problem is not the change itself, but the inadequate approaches for dealing with changes.

In order to ensure better and more efficient requirements change management, researchers and practitioners focus their attention on identifying the need for a change, assessing the impact of a change on other system artifacts and carrying out the change propagation to ensure co-evolution. In the remainder of this section we present the current work with regard to these activities.

Identifying the cause of requirements changes

A first step toward adequate requirements change management is to identify the cause of a change. According to Christel and Kang [CK92], factors that trigger changes in requirements are constant change in users' needs, disagreement among customers or stakeholders on requirements and changes in organization policies. Other studies found that developers' lack of domain application knowledge, hardware changes and evolution of competitors' products in terms of used technologies, contribute to frequent requirements changes [CKI88], [CF96].

Nurmuliani et al. [NZF04] presented a taxonomy related to the causes of requirements change. They identified the following eleven causes: defect fixing, missing requirements, functionality enhancement, product strategy, design improvement, scope reduction, redundant functionality, obsolete functionality, erroneous requirements, resolving conflicts and clarifying requirements. Based on this taxonomy, Saher et al.[SBG17] classified the reason of change into three main categories, i.e., mutable, emergent and consequential.

In [HBCG16] (see Chapter 2), we investigated the reasons for changes in requirements and confirmed many of the already identified issues. In addition, we found that changes of requirements are mainly triggered by the inability to implement the specified requirements due to underestimated complexity of requirements.

Change impact analysis

There are several hundred studies which are concerned with change impact analysis (CIA) [Leh11]. Bohner and Arnold [BA96] define software CIA as *"identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change"*. Bohner [Boh96] classified CIA techniques into two main categories: dependency analysis and traceability analysis. Traceability analysis is documentation oriented and performed at the requirement and design level, while dependency analysis concerns the automatic detection of dependency information at the source code level [AAA+15]. Therefore in the following text we concentrate on studies about traceability analysis and automated approaches for change impact analysis between requirements and other artifacts, that employ IR or NLP concepts.

Antoniol et al. [ACC+02] propose a method for an impact analysis between artifacts expressed in informal, free text and source code. They apply both a probabilistic and a vector space information retrieval model to trace C++ source code onto manual pages and Java code to functional requirements.

In order to support CIA, Hayes et al. present RETRO [HDS+07], a requirements traceability recovery tool that employs different IR methods, such as Vector space retrieval, and user relevance feedback. Their tool is a basis for other approaches that support change impact analysis between requirements and other software artifacts. For instance, by using RETRO, Ben Charrada et al. [CKG12] identify impacted requirements based on source code changes.

Arora et al. [ASG+15a] provide an approach for CIA for natural language requirements. They support inter-requirements CIA by calculating a normalized matching score based on propagation conditions. What distinguishes this approach from other NLP approaches for CIA is that it automatically detects and takes into account the phrasal structure of requirements statements and not only standalone words or characters. The approach is implemented in a prototype tool called NARCIA (Natural Language Requirements Change Impact Analyzer) [ASG+15b].

Borg et al. [BWRR17] introduce ImpRec, a recommendation system for CIA between artifacts written in natural language. ImpRec builds on assisted tracing, using information retrieval solutions and mining software repositories to recommend software artifacts written in natural language that are potentially impacted by changes in issue reports. The approach calculates similarity between the issue reports and recommends the potentially impacted artifacts for incoming issue reports.

Although some of these approaches can be applied between requirements and acceptance tests, none of them originally provides methods for automated CIA between such documents, despite the fact that aligning requirements and acceptance tests is of particular interest for industry and needs more automation. More details about requirements and tests alignment are described in Subsection 1.1.3.

Change propagation and co-evolution

There is limited research with respect to change propagation, especially in the context of natural language software artifacts. Yskout et al.

[YSJ10] present an approach for generating guidance on how to adapt architecture artifacts when requirements change by using change patterns. However, in their work all artifacts and change patterns are described in model-driven terms and not in natural language.

The aim of the work of Han [Han97] is to support developers with a framework for change management in the software engineering environment [WH94] to enable impact analysis and change propagation. The proposed impact analysis approach is a combination of automatic application of codified rules and interactive confirmation of potential impacts. It is applicable to code and non-code artifacts, but only in the in the context of a (generic) software engineering environment. The approach is not designed to be applied to the artifacts outside the application context, such as requirements or acceptance test documents.

Ben Charrada et al. [BCKG15] present an approach for identifying outdated requirements based on source code changes. They first identify changes applied to the source code, extract the keywords describing the changes and then trace these keywords to the requirements specification in order to identify not only which requirements documents are affected by a change, but also the parts of the requirements document that need to be adapted.

One of the shortcomings of the existing approaches is that they require certain formalization in specifying software artifacts, so that changes can be correctly propagated from one artifact to another. Moreover, the approaches do not consider acceptance tests, but rather focus on change propagation between requirements, source code or

architecture artifacts. In our work, we aim at ensuring co-evolution of requirements and their acceptance tests. Moreover, our approach does not require any formal syntax for specifying requirements and acceptance tests.

1.1.2 NLP research in requirements engineering

Since the early 90's researches have studied the role of NLP in requirements engineering [Rya93]. Berry et al. [BGST12] classified NLP tools into the following categories:

1. tools to find defects and deviations from good practice in natural language requirements documents;
2. tools to generate models from natural language descriptions;
3. tools to infer trace links between requirements among themselves or between requirements and other software artifacts;
4. tools to identify the key abstractions from natural language documents.

We previously explained how NLP can be used to support more efficient requirements change management via tools and methods for change impact analysis [ASG+15a], [BWRR17] and tracing [HDS+07], thus describing the tools from the third category in the classification of Berry et al. [BGST12]. In this subsection we provide an overview of the existing methods and tools that belong to the remaining categories.

NLP methods to improve the quality of requirements specifications and ease the communication between stakeholders

In their online market research [LMP04], Mich et al. assessed the demand for developing a NLP-based software engineering tool for analyzing documents written in natural language. The results of the study show that requirements documents are in most of the cases provided by customers and written in unstructured natural language. Therefore, the authors emphasize the importance of using linguistic techniques and tools to support requirements analysis.

Lucassen et al. [LDvdWB16] provide an Automatic Quality User Story Artisan framework (AQUSA). AQUSA uses NLP techniques to detect quality defects in user stories and to suggest possible remedies. They extend their work by developing an approach for extracting conceptual models from natural language requirements, called Visual Narrator [LRD+17]. The authors argue that extracting models from requirements helps to identify dependencies, redundancies and conflicts between requirements and, therefore, improves their quality. Moreover, together with other researchers (e.g. [HG03], [DM06]), they claim that this concept contributes to easier communication between stakeholders.

Rosadini et al. [RFG+17] investigated to what extent NLP can be practically applied to detect defects in the requirements documents. They found that, although human review cannot be replaced, NLP support can help in reducing the workload by prioritizing the requirements to be manually analyzed for defects. In addition, they found that NLP

can be used to check for defects left behind after a manual analysis has been performed, which is originally suggested by Berry et al. [BGST12].

In order to decrease ambiguities and inconsistencies in requirements and avoid misinterpretations, researchers used NLP methods to automatically derive formal UML models from natural language requirements, such as class diagrams [IA10], [MP12], activity diagrams [SGB14], state diagrams [MGND17] and others. These formal models are then used to communicate requirements among different stakeholders or to derive other models in an automated way. For instance, they are used for an automated creation of test cases from natural language requirements.

Generating tests from models obtained from natural language descriptions

Ryser and Glinz derive test cases from use cases, by firstly transforming unstructured use cases in natural language to statecharts [RG+00]. Similarly, Riebisch et al. derive test cases by using state diagrams [RPG03]. Hasling et al. use activity diagrams [HGB08] for this purpose. Other approaches for deriving tests from requirements in an automated way do not require additional behavioral modeling, but they require manual intervention by the testers for test generation, such as provision of input test data [dSJV12] or additional level of formalism when specifying requirements [dCdMBH15], [Heu01].

More recent approaches avoid additional behavioral modeling and use restricted use case modelling (RUCM) [YBL13] instead. For instance,

Wang et al. [WPG+15] use RUCM to automatically derive test cases from requirements written in free-form natural language. Other approaches use requirements specified in a Controlled Natural Language, such as SysReq CNL [dCdMBH15]. For example, NAT2TEST [CBC+15] is an approach and a tool for automated generation of test cases from natural language requirements specified in SysReq CNL.

Identifying the key abstractions from natural language documents

In order to support analysts to gain understanding of an unfamiliar domain, researchers implemented approaches for identifying the key abstractions from documents written in natural language. For instance, Gacitua et al. present their "Relevance driven abstraction identification" method (RAI) [GSG10], which identifies single- and multi-word abstractions, i.e. terms that have a particular significance in a given domain, in order to facilitate learning the key concepts of a new domain. Goldin et al. developed AbstFinder [GB94], an approach for finding abstractions in natural language text in order to support requirements elicitation.

We see that NLP is used for various purposes in requirements engineering. In our work, we use NLP for enabling change propagation between requirements and acceptance tests by generating guidance in natural language.

1.1.3 On aligning requirements engineering and testing activities

The alignment of requirements engineering and testing activities (RET) refers to linking information between people and/or documentation using mechanisms of varying formalism and complexity [UFG14]. For instance, methods for ensuring RET may be applied on organization, process or documentation level. In further text we explain the existing methods for ensuring RET and the most common challenges.

A proposed way for ensuring RET is to involve testing roles in requirements activities and change management process [BRB+14], [UKKD08]. Similarly, Fogelström and Gorschek [DFG07] found that involving testers as reviewers of requirements increases the interaction with requirements engineers and can improve the overall quality of the requirements, thereby supporting RET. Although clear benefits of involving test engineers in requirements-related activities are recognized, this practice is not always applied. The reason is unavailability of requirements and test engineers due to other assignments and distributed development [UKKD08].

One of the methods for aligning requirements and acceptance tests documents is an automated generation of traces between software artifacts, including requirements and acceptance tests. For instance, Lucia et al. [LFOT07], [DLFOT04] developed a traceability recovery tool based on Latent Semantic Indexing (LSI), Lormans et al. [LVD06] assessed how LSI can be used for reconstructing requirements traceability in design and test, while Cleland-Huang et al. [CHCC03] implemented an event-based tracing approach.

Another method for achieving RET is an automated derivation of acceptance tests from natural language requirements. Escalona et al. [EGM+11] assessed twenty-four approaches for automatic generation of test cases from functional requirements written in natural language, that were developed between 1988 and 2008. They conclude that none of the approaches closes the problem of generating functional test cases automatically in a satisfactory way and suggest the usage of standards for specifying documents and more formal methods to describe the process of generating tests from requirements. We described some of these approaches in the previous subsection (see NLP research in requirements engineering).

In general, using system level requirements to derive acceptance test cases imply the following challenges: requirements and the test cases may (1) differ in level of abstraction, (2) make use of different ontologies and (3) be written by and for people with different competencies [DEW+18]. In order to mitigate these challenges, researchers rely on transforming natural language requirements into formal representations and base their work on such formally expressed requirements.

Finally, in order to ensure RET, some researchers propose a usage of acceptance tests as requirements [MMC06], [MRM04], [BB17], [HS12]. This is especially accepted in agile communities, as this practice requires good, continuous communication [BB17].

The role of communication in RET

Several researchers emphasized the importance of communication for achieving RET [BMC+14], [BRB+14], [SLE+10], [LB14], [UKKD08]. For

instance, Bjarnasson et al. [BRB+14] identified that human aspects, such as cooperation and communication, are a critical basis for the alignment. Similarly, Sabaliauskaite et al. [SLE+10] investigated the challenges in RET caused by poor communication and inadequate tool support. Uusitalo et al. [UKKD08] emphasize the importance of implementing strong person-to-person links between requirement and test engineers. Moreover, Karthasu et al. [KIK+10] define communication as "the heart of agile software development".

Despite the identified importance of communication, the studies explain that the challenges in achieving successful communication are still present due to distribution of teams within organizations, poor documentation quality, domain knowledge of different stakeholders, an inadequate tool support or usage, etc.

1.1.4 State of the Art and Thesis Contribution

There is a significant amount of work in terms of requirements change management and alignment of requirements and acceptance tests, either through best practices or an automation of processes. However, there is still a lack of research on how to handle the changes and propagate them from requirements to acceptance tests.

There is a promising body of research on identifying affected documents based on changes in requirements. We shed a light on some of the existing approaches that use NLP and IR techniques to provide such links. We used two of the approaches, NARCIA and ImpRec to

assess to what extent they are able to correctly identify affected acceptance tests based on changes in the related requirements. In such a way we investigated whether the approaches can be used for CIA in GuideGen. The experiment and detailed results are presented in Chapter 6.

As previously shown, the current research on automated traceability for change impact analysis to a large extent answers the question of identifying *what* documents should be adapted when requirements change. However, research on *how* the affected documents should be adapted based on the changes in requirements is still missing.

In our work, we aim at filling this gap by providing concrete suggestions in natural language about how to adapt acceptance tests based on changes applied to requirements. In order to provide such suggestions, we rely on the existing research and assume that traceability links between requirements and acceptance tests exist. Our approach does not fully automate the change propagation between requirements and acceptance tests. Instead, it supports test engineers in making decisions on how to adapt acceptance tests, without necessarily communicating to requirements engineers nor manually comparing versions of a changed requirement.

Researchers point out that communication plays an important role in achieving requirements and test alignment. However, this is still a challenging task. With automatically generated guidance, e-mails and warnings about mismatches in artifacts, we aim at facilitating communication between different stakeholders, primarily between requirements and test engineers.

1.2 Research Goal and Questions

As motivated by the knowledge gap described in Section 1.1.4, we define our research goal as follows:

Thesis goal

Support requirements engineers and testers in maintaining the alignment between requirements and acceptance test documents when systems evolve.

In order to deliver an effective method for keeping requirements and acceptance tests aligned and up-to-date, it is essential to first investigate how requirements and acceptance test documentation is currently maintained in practice, what methods and tools are used and what challenges are encountered. Thus, our first research question is an exploratory knowledge question:

Research question 1 (exploratory)

RQ1. How are requirements and acceptance test documents currently maintained in practice?

By assessing the current state of practice and the faced challenges, we were able to better understand the current needs of practitioners with respect to requirements and acceptance tests management. We found a need for providing a novel method that can facilitate requirements and acceptance tests change management in an automated way. Therefore, we derived the following design research question:

Research question 2 (design)

RQ2. What is an effective tool-based approach for keeping requirements and acceptance tests consistent when system requirements evolve?

To evaluate how our approach meets the thesis goal, we perform two evaluation tasks. The first evaluation assesses how well the approach performs in terms of generating correct, complete and intelligible guidance. With the second evaluation we assess the usefulness and applicability of the approach. Hence, we derive the following two research questions:

Research question 3 (evaluation)

RQ3. How successful is our approach in generating natural language suggestions for adapting acceptance tests based on changes in requirements?

Research question 4 (evaluation)

RQ4. How useful and applicable is our approach for real-world projects and organizations?

In order to measure the success of the approach in generating guidance, we need to assess to what extent the generated suggestions are correct in terms of proposed actions, complete, understandable and compliant with the affected acceptance tests. To provide suggestions that are correct in terms of actions, the approach firstly needs to be

able to correctly identify the relevant change patterns applied to a requirement. Secondly, the rules for generating suggestions about how to adapt affected acceptance tests need to cover the relevant change patterns. Finally, the actions proposed by the approach must comply with the current content of the acceptance tests. Therefore, we further divide RQ3 into the following three sub-questions:

- RQ3.1: How successful is the approach in identifying the relevant change patterns applied to a requirement?
- RQ3.2: To what extent do the defined rules for generating suggestions cover the possible change patterns applied to a requirement?
- RQ3.3: To what extent are the generated suggestions applicable to the affected acceptance tests?

Since existing approaches for change impact analysis show promising results, as described in Section 1.1.1, in our work we primarily focus on *change propagation* between requirements and acceptance tests rather than on the change impact itself. However, as a requirement can have more than one related acceptance test, the approach should be able to firstly identify which acceptance tests are affected by a change before generating concrete guidance about how to handle the change. Therefore, we derive our fifth research question:

Research question 5 (exploratory)

RQ5. How well do the existing approaches for change impact analysis perform when applied between requirements and acceptance tests?

We answered our research questions in an iterative manner. The first four questions are fully answered in this thesis, while for RQ5 we present preliminary results, which provide the basis for our future work. In the following section we describe the research methodology applied to answer the questions.

1.3 Research Methodology

We choose a pragmatic approach for our research [CC17], as our main focus is to investigate *what* current issues practitioners are facing with regard to requirements and acceptance test documentation management and *how* these issues can be solved. By receiving feedback and further investigating challenges faced in industry, we kept our research practice-oriented and, therefore, we chose the pragmatic approach [ESSD08].

In our research we used the methodology inspired by Wieringa and Heerkens [WH06]. We started our research with investigating the problem (RQ1). Based on the obtained findings we designed the conceptual solution (RQ2) which was then implemented in a tool prototype. We iteratively evaluated and evolved our approach to get early feedback on our ideas and the tool prototype. In particular, as soon as we implemented the first prototype, we evaluated it by using real-world data obtained from three companies (RQ3). We then showed the prototype to seven practitioners from these companies and performed a further assessment of the practitioners' needs and potential improvements.

Based on the results form the first evaluation, we enhanced GuideGen and performed the second evaluation on usefulness and applicability (RQ4). Similarly to the previous evaluation, this evaluation enabled us to investigate further issues with the approach and the tool and elicit more requirements from practitioners. The second evaluation and redesign of our tool opened the new challenges that need to be further investigated (RQ5). An overview of the methodology applied in this research is presented in Figure 1.1.

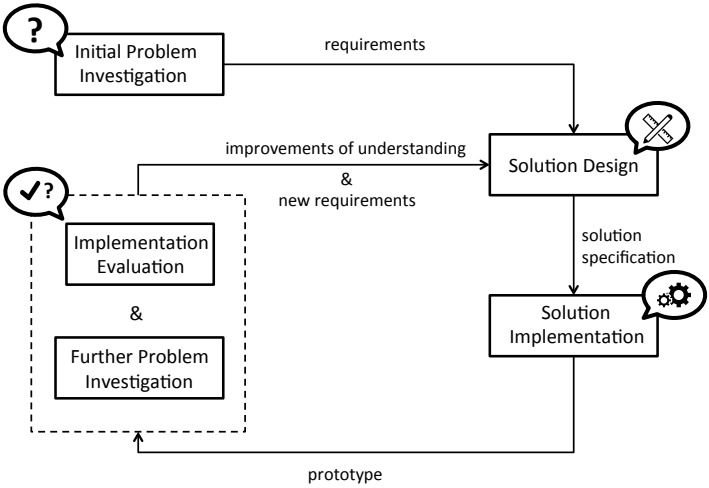


Figure 1.1: Our research methodology, adapted after [WH06].

1.4 Overview of GuideGen

In this thesis we contribute an approach for supporting requirements engineers and testers in keeping requirements and and acceptance tests aligned via automatically generated guidance. The name GuideGen stands for both: the conceptual approach and its proof of concept implementation. In this section we first explain the general operation context and then present two typical scenarios performed when working with GuideGen .

1.4.1 Operation context

Software-based systems constantly evolve, posing a challenge of maintaining software artifacts according to the newly introduced changes and keeping them consistent among each other. In order to facilitate the management of software artifacts, practitioners seek for appropriate tool support. In particular, among many tools for documentation maintenance and change management, several tools gained popularity and are regularly used in industry. For instance, Attlasian tools, such as Jira [jir] and Confluence [con], are currently the leading tools on the market. Popular tools for test change management are TestLink [tes] or QAComplete [qac]. Microsoft Word and Excel are still used in practice for documenting and managing requirements and acceptance tests, although they do not provide features tailored for effective change management.

All these tools are either requirements- or test-oriented and links between them are maintained manually. The documents are traced and

linked via IDs that must be inserted manually. This is error prone and time consuming and, therefore, often neglected by companies, which often results in having no links between documents at all. Automated approaches proposed by researchers and presented in Section 1.1.1 are not commonly applied in industry.

To our knowledge, TFS [tfs] is the only practically used tool that manages requirements and acceptance tests simultaneously and does not require manual linking between the documents. This tool adopts the concept of deriving acceptance tests from the existing requirements. In fact, the tests cannot be created independently without prior specification of requirements. We implement the same concept in GuideGen, as this is the common way of specifying acceptance tests [HKS97], [MSB11], [HBCG16]. Moreover, this concept provides traces between requirements and their tests without any additional effort.

Although TFS supports change impact analysis by providing features for maintaining links between requirements and acceptance tests, it does not provide support in terms of handling the change. GuideGen goes beyond the existing approaches by providing concrete guidance about how to adapt acceptance tests based on changes applied to their corresponding requirements.

1.4.2 GuideGen Scenarios

In this subsection we explain the two main scenarios performed when working with GuideGen: inserting new requirements and acceptance tests into the system and managing changes applied to a requirement.

Scenario 1. Insertion of new requirements and acceptance tests into GuideGen

There are two ways to insert new documents into GuideGen: 1. uploading requirements and acceptance tests from an external excel file and 2. adding new requirements and acceptance tests through the GuideGen user interface, as shown in Figure 1.2.

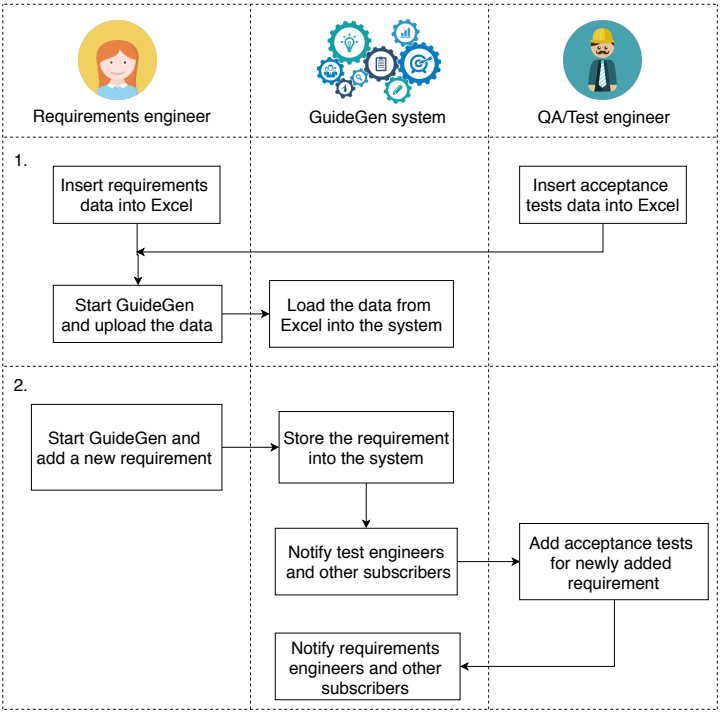


Figure 1.2: Uploading/adding new requirements and acceptance tests into GuideGen.

When uploading requirements and acceptance tests from an external document, the data must be provided according to the given template[1]. Currently, only import from an Excel file is possible, but this can be customized for other formats without a large effort. In this case, when data are uploaded from Excel, a requirements engineer and a test engineer separately insert the data into one shared Excel file and, once the insertion is complete, the requirements engineer uploads the data through the GuideGen application. This way of inserting new data into GuideGen requires prior communication between requirements and test engineers, when preparing the Excel file to be uploaded.

Another way is to insert new data through the GuideGen application. In such a way, no additional communication between requirements and test engineers is needed, as GuideGen notifies subscribers about every insert, via automatically generated e-mails.

Scenario 2. Change Management with GuideGen

In this subsection we explain how changes applied to requirements are managed and propagated to their related acceptance tests. Further, we explain the concepts used in GuideGen for managing the changes. Figure 1.3 presents the change management scenario with GuideGen.

As soon as a requirements engineer applies changes to a requirement, GuideGen analyzes the changes and generates guidance about how

[1]The template with the examples of requirements and their tests can be downloaded from the following Github repository: https://github.com/hotomski/guidegen

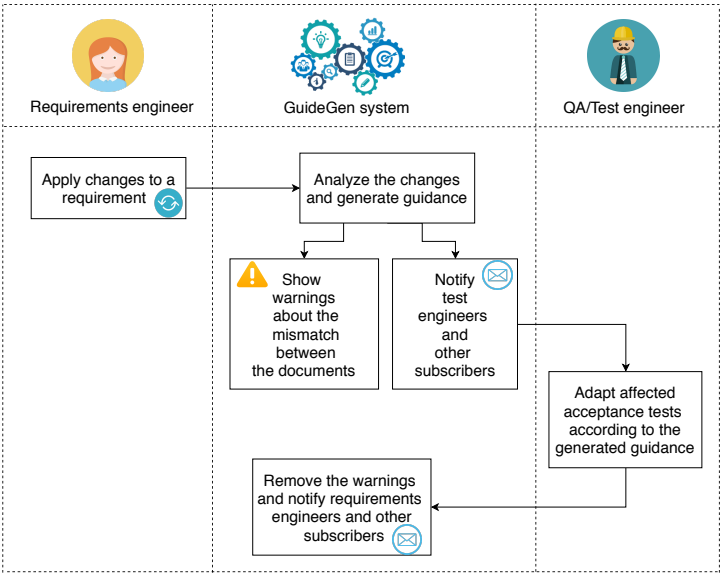


Figure 1.3: Managing changes with GuideGen.

the acceptance tests associated to that requirement should be adapted. The system then sends a notification e-mail to test engineers and other subscribers, such as product owners, or developers. In addition to the e-mail, GuideGen shows a warning sign next to the affected test ID. This warning sign indicates that there is a mismatch between the modified requirement and its related acceptance test. The mismatch represents all the differences introduced by changes in the requirement that were not applied in the affected acceptance tests, such as additions or deletions of sentences or word classes that cause changes in acceptance tests (e.g. nouns, verbs, numerical values, etc.). By showing the warning sign, we ensure that all stakeholders are aware

of mismatches through the application. As soon as the test engineer applies the suggested changes to the affected test case, warning sign is automatically removed and subscribers are notified about the adaption.

When more than one acceptance test is related to a requirement, GuideGen currently generates one, generic guidance for all of them. In order to identify which of the tests are actually affected by a change, an automated change impact analysis is needed.

We already investigated whether the existing approaches for an automated change impact analysis, explained previously in Section 1.1.4, can be used in GuideGen and we got to encouraging results. By incorporating one of the approaches, GuideGen could identify which of the related tests are affected and show the warnings and suggestions only for those tests. This will be the subject of future work.

The main concepts behind the change management scenario.

When changes in a requirement occur, GuideGen identifies relevant change patterns applied to the requirement, generates guidance about how to adapt the related acceptance tests and notifies the subscribed parties about the changes and the generated guidance, as presented in Figure 1.4.

Identifying relevant change patterns. When a requirements engineer applies changes to a requirement, GuideGen compares the old and the new version of the requirement and identifies sentences that have been added, deleted and modified. For the modified sentences GuideGen further identifies the exact words that have been added, deleted

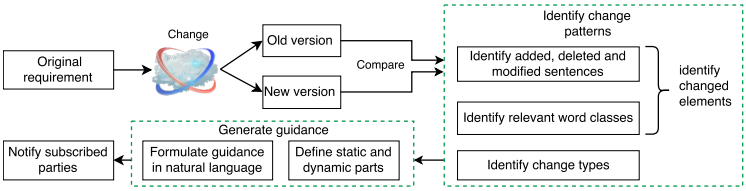


Figure 1.4: Managing changes with GuideGen.

or modified. When all the change patterns are identified, GuideGen determines which of them are relevant. Relevant change patterns are the ones that require the adaptation of the affected acceptance tests and only those are processed further.

Generating guidance. For every relevant change pattern, there is a rule in GuideGen describing how to formulate the static and dynamic parts of the suggestion that GuideGen generates for the corresponding change in a requirement.

Static parts describe actions that should be performed by testers in order to keep the acceptance tests consistent with the changed requirement. For instance, when a whole sentence is added, the following static part is formulated: *"Add new steps to verify that"*.

Dynamic parts fill in the gaps between static parts and describe the specific actions that are being tested within the test case (e.g. *"users can log in by using their phone"*). By combining the static and dynamic parts we formulate suggestions about how to adapt the affected test case (e.g. *"Add new steps which verify that users can log in by using their phone"*). We derived twenty-five rules for generating suggestions, which will be presented later in Section 6.3.

Notifying subscribed parties. GuideGen sends the generated suggestions to the subscribed parties via e-mail. The e-mail contains data about the affected tests that should be adapted, the list of generated suggestions and the summary of changes applied to the requirement. In addition, testers can find the generated suggestions in the application itself. The suggestions are shown in the form for editing the affected test case. Furthermore, warnings shown in the application indicate that there is a mismatch between requirements and their tests.

1.5 Roadmap and Chapter Summary

This thesis consists of five peer-reviewed scientific articles, presented in Chapters 2-6, which contribute to the achievement of the thesis goal. In the remainder of this section we describe the main contributions of each of the articles and map them to the research questions described in Section 1.2. In Figure 1.5 we illustrate the roadmap of this thesis, showing the relation between thesis chapters, research questions, research methodology and scientific articles.

1.5.1 Chapter 2: Writing and Managing Requirements and Acceptance Test Documents in Practice

Chapter 2 represents the *initial problem investigation* step in our research methodology and answers the first research question (RQ1). In

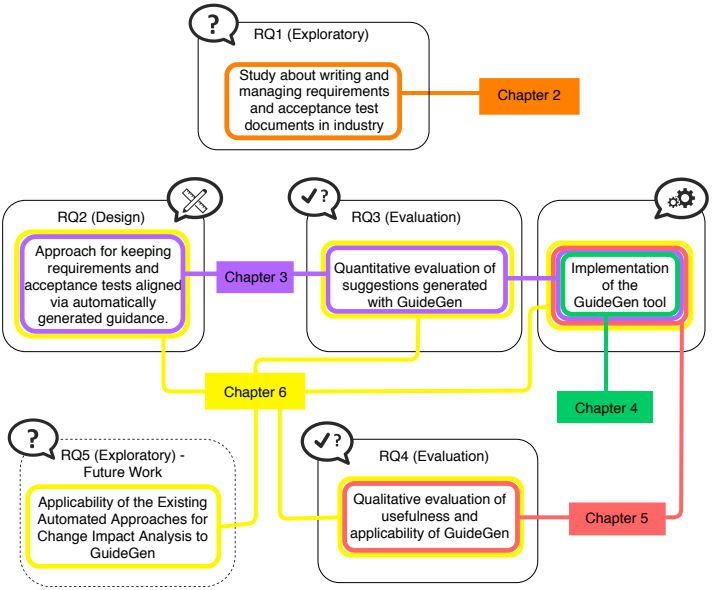


Figure 1.5: The roadmap of the thesis.

Chapter 2, we describe an exploratory study, conducted with twenty practitioners from fifteen companies. The study investigates how practitioners write and maintain requirements and acceptance test documentation and what challenges they encounter. In addition, we investigated to what extent practitioners are satisfied with the current tools for managing requirements and acceptance test documentation and what features they envision for more efficient documentation management. Therefore, the main contribution of this work are revealed challenges and identified features, needed for enabling more efficient documentation management.

The results of this study show that there are still many challenges encountered when writing and maintaining requirements and acceptance test documents. For instance, test engineers are still not included in requirements engineering activities and additional, preferably verbal, communication is requested in order to align requirements and testing activities. Furthermore, we found that acceptance tests are not regularly maintained, which, for instance, results in reporting bugs for actual features.

With regard to the current tool support, the study showed that the currently used tools do not provide much automation for maintaining trace links and changes.

1.5.2 Chapter 3: Keeping Evolving Requirements and Acceptance Tests Aligned with Automatically Generated Guidance

Chapter 3 represents *solution design and implementation evaluation with further problem investigation* in our research methodology. In Chapter 3, we describe the conceptual solution, all the algorithms and methods used in the initial GuideGen implementation and define rules for generating suggestions, thus answering our RQ2.

We evaluated the implemented approach and the prototype tool by using three datasets from three companies. For 262 non-trivial changes of requirements GuideGen generates suggestions that are further evaluated by seven practitioners from the companies. The goal is to assess to what extent suggestions, generated by GuideGen, are correct, complete and understandable. The assessment is performed by practitioners who characterized each of the generated suggestions as correct or incorrect according to the seven criteria. Therefore, with the study described in Chapter 3 we also answer our RQ3 and its subquestions.

The study also includes a short qualitative assessment of GuideGen, aiming at *further investigating* the possible improvements of the approach and the tool. Based on feedback obtained from the seven practitioners, we improved GuideGen accordingly. The improved version of the GuideGen tool is described in Chapter 4 and further evaluated in our qualitative study, which is presented in Chapter 5.

1.5.3 Chapter 4: GuideGen: a Tool for Keeping Requirements and Acceptance Tests Aligned

Chapter 4 of this thesis presents the GuideGen tool, including a short demo video[2]. In this publication we present the typical scenarios performed when using GuideGen as a documentation management tool. The goal is to present the principal features of GuideGen and its user interface, so that the audience can get the first impression of how it would be to work with the tool and what benefits it brings.

This chapter represents the *implementation* in our research methodology. Although Chapter 4 does not cover any research questions, it is an essential part of this thesis, as it presents the practical implementation of our conceptual solution.

1.5.4 Chapter 5: Usefulness of GuideGen and its Applicability to Real-World Projects

In Chapter 5, we describe our interview-based study, conducted with twenty-three industrial practitioners from ten companies. Every trial in this study consists of three parts: (1) introductory presentation, in which we presented the GuideGen approach and showed the demo video; (2) tool trial, in which practitioners used examples from their current projects to try the GuideGen tool; and (3) interview with practitioners, in which we asked about the practitioners' opinion towards the general usefulness and applicability of the approach and the tool.

[2] GuideGen demo video: https://vimeo.com/254865530

The goal of this study is to investigate to what extent are automatically generated suggestions useful for practitioners and whether they perceive the proposed way of communicating requirements changes as beneficial. The study shows that having suggestions on how to adapt the acceptance tests based on the changes in requirements is superior to only being informed about the changes in a requirement. Communicating changes via automatically generated e-mails and warnings is seen as a positive aspect of the approach, although some practitioners expressed concerns about a potential flood of notifications when changes occur frequently.

With this qualitative study we reveal the benefits of GuideGen identified by the practitioners and investigate potential improvements in order to define a baseline for our future work. Therefore, Chapter 5 presents *implementation evaluation and further problem investigation* in our methodology. With this qualitative study we answer our RQ4.

1.5.5 Chapter 6: GuideGen Overview and a Step Towards an Automated Change Impact Analysis

In Chapter 6, we provide a comprehensive GuideGen overview. Based on the identified improvements in the qualitative study described in Chapter 5, we further adapted GuideGen, so that it potentially becomes applicable to wider range of projects. In Chapter 6 we describe all the adaptations, provide a concise explanation of the rules for generating suggestions and explain technical details and choices we made when implementing the approach, thus covering our RQ2-RQ4.

In addition, Chapter 6 includes details about an experiment we conducted in order to make a first step towards achieving change impact analysis with GuideGen. We applied the existing approaches for change impact analysis between textual documents, NARCIA [ASG+15b] and ImpRec [BWRR17], to a set of requirements that have more than one acceptance tests related. The data set is obtained from an industrial project, provided by a medium-size company.

The goal of the experiment is to assess how well the existing approaches perform when applied between requirements and acceptance tests, thus revealing whether these approaches can be used for identifying affected acceptance tests based on changes in their requirements. Therefore, Chapter 6 discusses potential future work and answers our RQ5.

1.6 Contributions

The contributions of this theses are twofold.

Firstly, in our exploratory study we analyzed *the state of practice with regard to requirements and acceptance tests documentation in industry*. We revealed the characteristics of the typical process of writing and maintaining the documents during the software evolution and identified the most frequent challenges that practitioners encounter. In such a way, we learned about the practitioners' needs towards more efficient documentation change management. In particular, we focused on finding an efficient method for keeping requirements and acceptance tests up-to-date and aligned.

Secondly, based on the lessons learned from the exploratory study we developed and evaluated *a novel approach for keeping requirements and acceptance tests consistent by using automatically generated guidance*. Therefore, making the following contributions:

- The conceptual solution of our approach, which relies on IR and NLP concepts in order to automatically generate guidance about how to adapt acceptance tests based on changes in unstructured, textual requirements;
- An implementation of the conceptual solution represented by the prototype GuideGen tool;
- Two evaluations of GuideGen - the first one shows how successful GuideGen is in generating concrete suggestions in natural language, while the second one shows how useful and applicable GuideGen is for real-world projects.

Chapter 2

Writing and Managing Requirements and Acceptance Test Documents in Practice

Original publication:
An Exploratory Study on Handling Requirements and Acceptance Test Documentation in Industry
S. Hotomski, E. Ben Charrada, and M. Glinz
24th IEEE International Requirements Engineering Conference (RE'16), pages 116–129, IEEE, 2016 ([HBCG16])

Abstract

With the emergence and spread of agile processes, the practices of writing and maintaining documentation have drastically changed in the last decade. In this work, we performed a qualitative study to explore the

current practices for managing two related types of software documentation: requirements and acceptance tests. We interviewed twenty practitioners from seventeen business units in fifteen companies to investigate the companies' practices for writing, maintaining and linking requirements and acceptance test documentation. The study yields interesting and partially unexpected results. For example, we had expected that tests would be more extensively documented than requirements, while we found a strong linear correlation between the number of requirements and tests in our sample. We also found that technical people are usually not involved in the requirements engineering activities, which often results in misunderstood or underestimated requirements. Acceptance tests are written, in many cases, based on requirements that are not necessarily detailed enough. Also, acceptance tests are not regularly maintained, which occasionally results in confusing features and bugs.

2.1 Introduction

Requirements engineering and testing are related activities that show much synergy. Linking the testing and requirements activities can benefit both sides and save money and time [Gra02]. For example, conceiving test scenarios concurrently with the elicitation and documentation of requirements helps uncover potential problems earlier, thus leading to better software quality [DC06].

Among the different types of tests, *acceptance tests* are those which are most related to requirements as they strive for *"comparing the program to its initial requirements and the current needs of its end users"* [MSB11]. Due to this strong relation, several advocates of agile software development even suggest to write requirements in the form of acceptance tests [MM08][Pug10].

In this work, we explore the current practices related to requirements engineering and acceptance testing.

Whenever we talk about tests, testing, or test documentation in the remainder of this paper, we always mean *acceptance* testing. The term "requirements" denotes *product* requirements in the context of this paper.

In contrast to previous studies on requirements and testing (e.g. [BRB+14], [UKKD08]) our study focuses specifically on the *documentation aspect* of the requirements and acceptance testing activities. Concretely, we investigated how the requirements documentation and the tests documentation are written, by whom they are written, in what format they

are specified and how useful they are for the requirements engineering and acceptance testing activities.

We also examined how the documentation is evolved and maintained. For instance, we looked for the main triggers for change in requirements and tests, the current update procedures and the factors that support or hinder the documentation update. The results of this paper are based on semi-structured interviews with twenty practitioners.

The paper is organized as follows. In Section 2.2 we describe the research methodology, including study design, research questions and threats to validity. Section 2.3 presents the key findings of our study, which are then discussed in Section 2.4. Section 2.5 presents related work. Section 2.6 concludes the paper with a summary and directions for future work.

2.2 Research Goal and Methodology

Our goal is to understand the current practice of requirements and acceptance test documentation management, as well as the challenges faced during this process. As a first step towards this goal, we conducted a qualitative exploratory study, using semi-structured interviews [8]. The interviews were based on a predefined interview instrument[1], composed of five parts. The first and the second part focus

[1] https://data.mendeley.com/datasets/szvyjpytyh/1

on the characterization of the interviewees and the companies, respectively. In the third part we investigate the company practices for writing requirements and test documentation. Next, we explore the practices related to the evolution and update of the documentation. Finally, we asked the participants about the characteristics of the perfect tool for supporting the documentation management. We also gave the participants the opportunity to freely talk about matters that they think could be related to the topic of the interview and which have not been addressed.

2.2.1 Research Questions

From our research goal, we derived the following two research questions:

RQ1. *How are requirements and test documents written in practice?* With this question, we aimed at exploring how requirements and test documents are written, who writes them and in which format they are specified. Besides, we looked into the factors that influence the documentation practices, such as the company organization, the used process models and the available tools.

RQ2. *How are requirements and acceptance tests updated and what difficulties are faced?* With this question, we aimed at examining how the requirements documentation and the test documentation are maintained and evolved. For example, we investigated what are the main triggers for change in requirements and in tests. We also explored the current documentation update practices as well as the factors that support or hinder the documentation update.

2.2.2 Study design

Preparation

Based on a review of the existing literature about the management and use of software documentation in practice, we shaped our research questions and created the interview instrument to answer the research questions. The interview instrument was validated and improved in two rounds: first it was reviewed and discussed within our research group. Second, we conducted two pilot interviews, one with a researcher from our group and one with a practitioner who was not involved in the study design.

Participants

For the selection of participants, we used a purposeful sampling strategy [Pat90]. The selected participants needed to satisfy the following two criteria: [C1] they work for a company in which requirements and test documentation exist in some form, [C2] they are knowledgeable about and/or directly involved in the requirements engineering and design processes and the test planning process.

We interviewed twenty practitioners from seventeen business units in fifteen companies, located in seven countries. The interviewees had between two and twelve years of experience in their respective fields.

Figure 2.1: Geographical distribution of companies

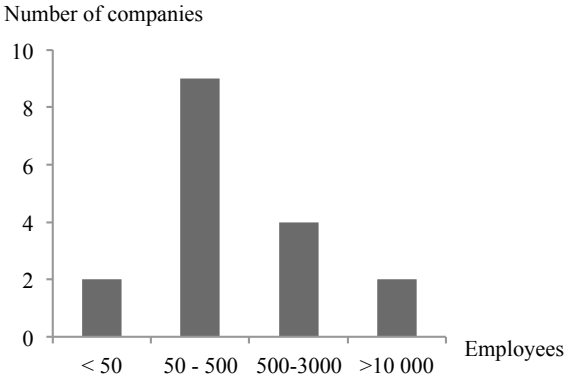


Figure 2.2: Size of the companies (number of employees)

The geographic distribution and the size of the companies are shown in Figure 2.1 and Figure 2.2, respectively.

Companies C4 and C7 both consist of two units that have significantly different characteristics in terms of the software process model applied, the team structure and the type of the produced software. In both companies we interviewed one representative from each unit separately. Therefore we treated these units as separate companies, named C4.a, C4.b and C7.a, C7.b, respectively. Thus, for simplicity, we report about a total of seventeen companies in the remainder of this paper.

We interviewed the participants individually, except for companies C3, C8 and C12, where we interviewed participants in pairs. Since these interviewees had different roles in their respective companies, we could obtain two complementary views of the practices in these companies. An overview of the participants and their companies is shown in Table 2.1.

We distinguish two types of companies: *product-oriented* companies develop market-driven software products, while *project-oriented* ones develop bespoke software for specific customers. The main differences between the two types are summarized in Table 2.2.

Only three of the interviewed companies use a waterfall software process model; all others employ different types of agile development models (see Figure 2.3).

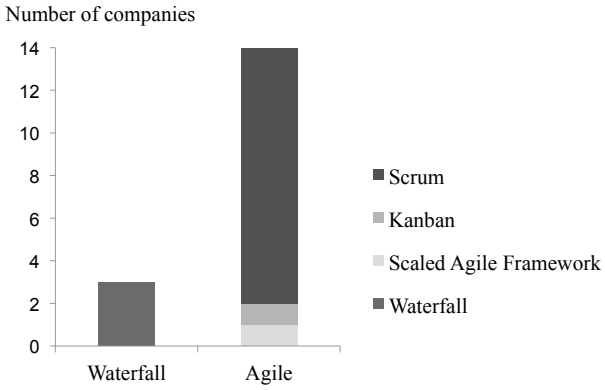


Figure 2.3: Software process models used

Table 2.1: Overview of the participants and their companies

Comp.	ID & role	Domain	Type	SPM
C1	P1 - Test developer	Outsourcing	project	S
C2	P2 - Test engineer	Games and technology	product	S
C3	P3 - Test developer	IT consulting,	project	S
	P4 - Prod. manager	outsourcing		
C4.a	P5 - Prod. manager	Electricity distribution,	product	SAF
C4.b	P6 - Test lead	automation management	project	W
C5	P7 - Test engineer	Technology services, custom application	product	S
C6	P8 - IT Consultant	ERP systems, ECM systems	project	W
C7.a	P9 - IT Consultant	Cloud storage services	product	S
C7.b	P10 - IT Consultant	Performance optimization services, mobile app development, outsourcing	project	K
C8	P11 - RE engineer, P12 - Software dev.	App for car sharing, business consulting	project	S
C9	P13 - Proj. manager	IT services in finance, insurance and media	project	S
C10	P14 - Business analyst	Assurance, consulting, financial, legal	project	S
C11	P15 - Test engineer	Storage platform and services, security	product	S
C12	P16 - Proj. manager, P17 - Test engineer	Medical and logistics software	project	S
C13	P18 - Business analyst	Health insurance	project	W
C14	P19 - QA/Test lead	Payment services	project	S
C15	P20 - Business analyst	Invoicing, marketing, customer care platforms	project	S

SPM - Software process model, S - Scrum, K - Kanban,
SAF - Scaled Agile Framework, W - Waterfall.

Table 2.2: Differences between product and project oriented companies

Product-oriented company	**Project-oriented company**
The software is developed for and driven by the market	Bespoke software is developed for a specific customer
The company specifies the requirements based on a market analysis	Requirements are elicited from the stakeholders
Developing the software is a continuous and iterative process	The development ends when the stakeholders' requirements are met and the product is delivered
Developing the software is a long-term process	Developing the software is a short-term process
The company owns the software	The customer owns the software

Data collection and analysis

The interviews were carried out between July and November 2015. All interviews were conducted by the first author. Face-to-face interviews were used whenever possible. In total we had ten face-to-face interviews and seven interviews via Skype. The interviews were conducted in English, except for two interviews, which were in Serbian because the interviewees felt more comfortable to use their native language.

The duration of the interviews was between 55 and 120 minutes. The face-to-face interviews were generally longer, because participants showed us their documentation. All interviews were audio recorded (with permission from the interviewees). For analyzing the data, the interviews were transcribed and coded. Serbian interviews were translated to English during transcription. We created an initial list of codes based on the interview questions and complemented it with codes that emerged during data analysis. We used the codes to group related answers, compare them to each other and derive hypotheses.

Threats to validity

As for every qualitative study, the validity of the findings is subject to several threats [WRH+12].

Below we explain the main design decisions we made with the aim of limiting the potential threats.

Construct validity is threatened if the answers from the participants do not accurately reflect the real practice. This could be due to the interviewees not feeling comfortable talking about certain topics or to the interviewer influencing the discussion. To limit these threats, we avoided judgement and evaluations during the discussions, we assured the interviewees about the anonymity of the study and we abstained from communicating our hypotheses to the participants.

In order to collect reliable data about the companies' practices, we only selected participants who are knowledgeable about requirements and testing procedures on the company level and not only within their team (criterion C2). When we interviewed participants from different units of the same company, we asked them not to talk about the interview to others in order to avoid bias. Unclear questions and misunderstandings between the interviewee and interviewers are also possible threats that cannot be completely ruled out. To limit these threats, we discussed the interview questions with RE researchers from our research group before conducting the interviews and conducted two pilot interviews. Thus we could revise the questions that turned out to be unclear or generated misunderstandings. We also tried re-explaining the questions differently to the interviewees whenever we thought that the participant misunderstood us.

However, reliability threats that relate to researcher bias cannot be completely ruled out, because the interviews and analysis were conducted by only one researcher.

External validity issues are related to the inability to generalize the results of the study beyond the studied companies. In order to achieve

reasonable generalizability, during the sampling, we selected companies that have different characteristics in terms of size, domain of activity, internal organization and location.

Internal validity issues appear if a causal relationship between treatment and outcome is wrongly established. Possible factors that could negatively impact the internal validity in our case are selection bias and interview instrument change. Although we started the selection of participants with our personal contacts, the network was soon spread with indirect contacts, thus decreasing the selection bias threat. All participants were contacted directly and none of them declined. This decreases the threat of having only participants who are interested in the topic. Regarding the interview instrument change, we evaluated the interview questions with two pilot interviews and revised them before starting the real data collection. So we avoided any changes to to the interview instrument during the actual interviews.

2.3 Key Findings

In this section, we present the key findings of our study, grouped by research question. As our our findings are not statistically representative, we formulate them in terms of hypotheses, for which we have evidence from our study data.

2.3.1 Writing the requirements and test documentation in practice - RQ1

Hypothesis H1.1

Technical people are not involved in the requirements engineering activities.

Except for one company (C3), where the requirements are provided by the client directly, all companies have the requirements specified by a person in a managerial position, such as project manager, product owner, business analyst or consultant. These managers may have a technical background, but their engagement is only managerial and they are not aware of technical details. Managers elicit, specify and maintain the requirements based on communication with stakeholders or their own, market-driven views. Engineers, who will be responsible for the development and testing of the product, are not involved in any of these activities. As managers do not always have the necessary technical knowledge about the project, this results in two major problems: (1) the requirements are often misunderstood and (2) their complexity is underestimated.

A few participants mentioned the amount of work as a possible reason for not involving technical people in the requirements engineering activities: *"One person cannot worry about what is needed and also how it will be implemented. That is just too much"* (P20). Additionally, in many companies, a strict separation of roles is applied, which limits the domains of responsibility of the practitioners to only one aspect

such as design, implementation or testing. This is especially true for participants who are involved in many projects.

The organizational hierarchy could also have a negative impact here: *"People do not want to ruin the hierarchy. Business people are on a higher level of hierarchy, even if they are not capable sometimes to make the right decisions. Giving the opportunity to the developers or testers to decide about features would bring less power to the business people and they are not willing to accept that"* (P3). In one company, finances are the problem. Since their customers are spread over different continents, *"it is costly that the whole team attends design meetings and this is why only business people are involved"* (P19).

Hypothesis H1.2

The format of requirements specifications depends on the software process applied.

The companies following a waterfall process document requirements in a traditional format: In C4.b and C6, the requirements are written in prose and stored as text documents. In C6, these documents contain not only the requirements, but also test cases. In C13, the requirements are specified as structured use cases: *"It is much easier to keep requirements up to date if they are not in prose text, but in form of use cases, because it is easier to spot the part that needs to be changed"* (P18).

In the companies employing an agile process, user stories [Lef10] are the most common way for documenting requirements. User stories

contain acceptance criteria, which are further used for writing the acceptance tests. Eight companies use user stories only, four combine user stories with sticky notes or index cards and two companies augment the user stories with detailed explanations. Sometimes, requirements are written on index cards or sticky notes only, which are then arranged on walls or tables to facilitate planning and discussion (C2). However, in none of the companies this was a standard format for requirements.

Most of the companies store and manage user stories in JIRA or Confluence. An overview of the tools and repositories used for managing and storing requirements and test documentation is given in Figure 2.4. Some companies use more than one tool for managing acceptance tests.

None of the agile companies produce a vision document. Companies C8 and C9 produce requirements documents that contain a detailed explanation of every requirement, in addition to user stories. However, these requirements documents do not contain information regarding the strategic intent of the product to be built and thus, these are no vision documents [Lef10]. In C9, more detailed requirements documentation is requested from the clients: *"Sometimes, clients are willing to apply agile methodologies in terms of continuous delivery of software and their involvement during all phases, but they insist on having more documentation in traditional form of prose text"* (P13). However, due to time limitations and difficulties to handle changes in an unstructured text, this document usually becomes outdated when the requirements change. In C8, well established procedures and habits result in having more detailed requirements documentation, which will be discussed later (see Hypothesis H2.5).

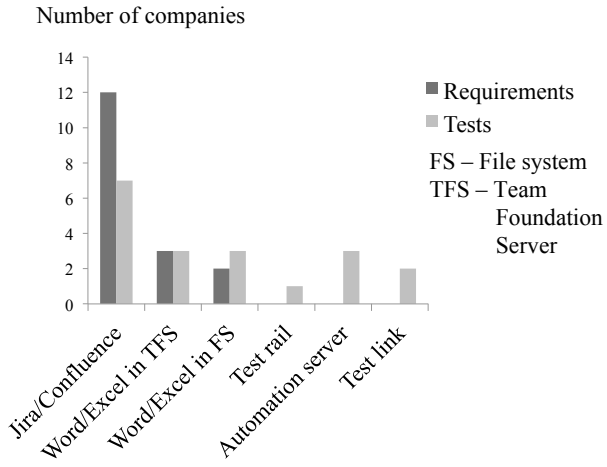


Figure 2.4: Repositories and tools for storing and managing requirements and acceptance test documentation in the interviewed companies

Hypothesis H1.3

There is a strong linear correlation between the number of requirements and acceptance tests.

In all companies, requirements are specified first and each test case is derived from the associated requirement. The companies aim at covering each requirement by at least one acceptance test case. However, the results show that the number of acceptance tests is only marginally larger than the number of requirements. In none of the companies, requirements are covered by more than two acceptance test cases. Moreover, in C3 and C6, the number of tests and requirements is reported to be equal: there is only one test case for each requirement.

Figure 2.5 illustrates the strong linear correlation between the number of requirements and acceptance tests per project, with an average ratio of 1:1.27. The estimates were given by the participants.

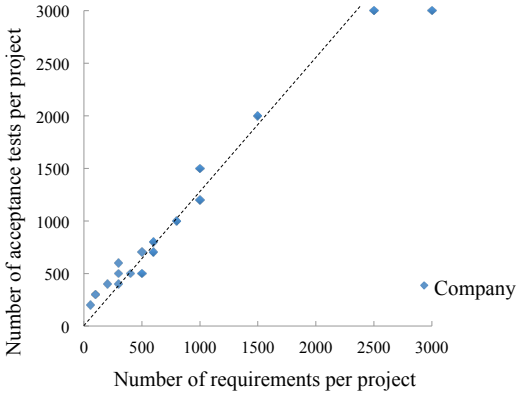


Figure 2.5: Approximate number of requirements and tests per project.The dotted line shows the average ratio.

Acceptance test-driven development (ATDD), is reported as a good practice by practitioners. Therefore, except for one company (C4.a), where the tests are written at the end of every sprint, all companies following an agile process write the acceptance tests before the source code, whenever possible. Tests would be written in parallel with the source code only in the case of time constraints. Writing the acceptance tests before the code is the first principle of ATDD. However, the second principle of ATDD, which states that requirements documentation should be specified in the form of acceptance tests [MM07], is not applied among the interviewed companies, because every test case covers one, separately written requirement.

Hypothesis H1.4

Test engineers write acceptance tests based on the requirements that are not necessarily complete.

According to Humble and Farley [HF10], testing is a cross-functional activity that involves the whole team and should be continuously done from the beginning of the project. However, our study shows that tests are usually specified only by a dedicated, technical role, such as test engineer or, sometimes, a developer (C7.a). The rest of the team is not included in this process. Only in companies C6 and C13 tests were specified by the same person who specified requirements. That person has a managerial role. In C10 the acceptance test were specified by the client.

When writing the test documentation, test engineers first refer to the requirements specification. Only if something is not clear, they communicate with managers or developers. However, participants admit that this additional communication is often needed: the test cases cannot be derived from the requirements documentation alone, because it is not detailed enough. *"We have requirements in form of user stories with acceptance criteria defined. Sometimes, this is not enough for me to write the test cases, so I need to communicate with developers or the product owner. They are not always easy to catch, since we are distributed and people are busy. Then, I just wait for the next weekly meeting"* (P7). Participants also said that sometimes this results in postponing deadlines.

Oral communication is not always the optimal solution. In fact, participants stated that having a more detailed requirements document would also help them in writing tests.

Hypothesis H1.5

Acceptance tests are mostly written manually, using the typical format.

Typical manual tests are written in natural language and contain the usual elements: *name, prerequisites, ID, description, steps (actions)* and *expected results*. Except for one company (C1), which uses more automated tests than manual ones, all companies either use manual tests only (11 companies) or manual tests combined with automated tests (C2, C4, C10).

The automated tests contain the scenario and the code related to this scenario via annotations. A scenario is written in natural language using the *Given-When-Then* template [Adz16]. Therefore, the test specification for automated tests does not differ much from the manual test specification in terms of structure and format:

- *Given* describes the state of the world before starting the behavior. It corresponds to *prerequisites* in manual tests.
- *When* describes the key action the user performs. It corresponds to *steps (actions)* in manual tests.
- *Then* describes the expected outcome of the behavior. It corresponds to *expected result* in manual tests.

Cucumber and Selenium are the main tools that companies use for creating and executing the automated tests. None of the companies uses tools for automatically generating tests.

Hypothesis H1.6

Mixed and centralized teams produce less documentation than distributed ones.

We found a relation between the amount of documentation and the team organization and structure. The structure of the teams differed much from one company to another. While in some companies teams are mixed, in others they are separate. By separate we mean that each team has a specific responsibility for the software, such as development, testing and QA, deployment or technical writing. Mixed teams include people responsible for various roles, or a representative for each of the roles. In distributed teams, the members are located in different cities or countries. In centralized teams, the members are in the same office or building.

In agile companies, teams are mostly centralized. In fact, in some companies, testers and developers were co-located, while the product owner and scrum master were distributed. In distributed teams, daily or weekly meetings are regularly held via tools for web conferencing, but both managers and engineers rely more on the existing documentation when they write or execute tests, create tasks or implement features. This usually results in these teams having more detailed documentation than co-located ones.

In mixed and centralized teams, people rely more on (informal) communication than in distributed or separate teams. In two agile companies, participants said that requirements are documented as user stories, but in some cases, the requirements documentation exists

only in the form of sketches, drawings on whiteboards or sticky notes: *"Sometimes we do not even have requirements in form of user stories. We capture only what is important in form of notes and drawings and discuss them during meetings, so we know how to test the features and developers know how to implement them. User stories are only a call for discussion. This is possible because we are all here; we communicate"* (P2). By "call for discussion", P2 meant that the user stories only trigger further communication and are used as a reminder of current sprint elements that need to be discussed during the meeting.

2.3.2 Updating the requirements and acceptance test documentation in practice - RQ2

Hypothesis H2.1

Changes of requirements are mainly triggered by the inability to implement the specified requirements.

When discussing the triggers for changing requirements, we found that the majority of the change requests that impact requirements did not come from external stakeholders (e.g., clients), but rather from the internal ones. In fact, as technical engineers were usually not involved in the elicitation of the requirements (see Hypothesis H1.1), the complexity of the requirements was frequently underestimated: *"Business people often see only where we should get, but not what it takes to get us there. They do not know if something is possible to implement and how."* (P12). Underestimated complexity often results

in the inability to implement the specified requirements. In nine companies, this inability was reported as the cause for more than 70% of the changes in requirements.

Internal change requests can also be triggered by changes in the used platforms, frameworks and tools. Sometimes, such changes in the infrastructure require a redesign of the software architecture and also the requirements, as reported in two companies.

Changes in requirements can also be requested by external stakeholders if they previously were not sure about what they need or if they changed their mind later on: *"At the beginning, clients sometimes cannot imagine how something will really look like or how it will really work and fit to the rest of the system."* (P19). Furthermore, changes may happen due to misunderstandings between the people eliciting the requirements and the external stakeholders.

The main triggers for changing requirements in the investigated companies are presented in Table 2.3. In these companies, the trigger contributes to more than 70% of the requirements changes. In C1, C3 and C15, internal and external triggers impact requirements equally. Therefore, these companies are not included in Table 2.3.

Hypothesis H2.2

Existing requirements are updated in waterfall-style projects while new user stories are created in agile ones.

Table 2.3: Main triggers for changing requirements documents in companies

Trigger type	Reason	Companies
Internal	Underestimated complexity of requirements	C2, C4.b, C5, C6, C7.a, C7.b, C8, C9, C11
	Infrastructure/architectural change	C4.a, C13
External	Clients are not sure about their needs at the beginning	C12, C10
	Clients change their mind about what they require or requirements are misunderstood	C14

In all three waterfall projects that we covered in the interviews, the requirements are usually kept up-to-date. In two of them, the participants claimed that requirements are always changed before the code is changed: *"We update the documentation first, and later the code. Otherwise there is no point to have any documentation if nobody can trust it. We put a lot of effort sometimes to track changes, but that is all part of the job"* (P6). In the third company, the requirements are updated after the code is changed, and mainly because the external stakeholders require it.

Agile companies handle the requirements differently. In fact, instead of updating existing requirements, they usually create new user stories whenever a change is required. Even if the change occurs in the middle

of a sprint, the procedure is the same: the work on the old user story is stopped, and a new user story is created, prioritized and put in the backlog. The new user story might then be assigned to the next sprint or not, based on its priority: *"After change and prioritization, that new user story is sometimes not developed for months, or even years. By then we will for sure forget what was the original requirement, but do we even care?"* (P5).

An exception is made in the case of minor changes, as no new user story is created: *"We will make new user stories only if the change is not, for example, changing the label from 'Email' to 'E-mail' or a similar cosmetic change. Otherwise, although we know this is not by the book, we will change the existing user story and continue the sprint, because it would take much more time to create a new user story in this case."* (P16). Similarly, P9 explained:*"We will create a new user story instead of updating the existing one only if the change is greater than 20%"* (P9).

Although most of the companies keep the old user stories in the backlog, a few companies do not. In fact, in three companies, the user stories are cut from the backlog into the sprint instead of being copied. When the story needs to be changed, a new one is created, placed in the backlog, and the one from the sprint is deleted. In one company (C12), user stories are copied from the backlog into the sprint, but deleted from both, backlog and sprint, when they are replaced by new ones.

Only in company C8 new users stories are linked to the old ones in the backlog, which is also the original requirement. The others reported

no need for linking the old and the new user stories: *"We do not need to know what was the requirement in the past. The only relevant reference point is the source code. It [the source code] is the requirement from the past and then we start from that reference point and make changes"* (P11).

Hypothesis H2.3

Acceptance test documentation is less frequently updated in teams with good communication.

The practices regarding the update of acceptance tests vary considerably among the companies. Some companies keep their acceptance tests regularly up-to-date by updating them either before the code is changed or in parallel with the code. However, about half of the participants mentioned that in their projects, the acceptance tests are only updated after the code is changed. For these projects, it also happens that the acceptance tests do not get updated. This is mainly because the testers rely much on the communication with the engineers and developers to run the tests. This is especially true for small and centralized teams where the testers gather information about the changes in meetings with the developers and managers, manually test the changed features, and often leave the test documentation unchanged.

Although relying on informal communication works rather well with small and centralized teams, sometimes this also leads to problems. For instance, when the change is not well communicated, a feature

could be mistaken for a bug: *"Sometimes we need a lot of time to realize that it is not a bug, but a feature. We need to communicate with developers and check with the product owner in order to be sure that the requirement has changed, which can take quite some time" (P19).*

Another cause for outdated test documentation is the use of exploratory testing [IML09], where testers mainly rely on their own experience and intuition in order to find weak areas and, therefore, do not use the test documentation much. Our study confirms the usage of exploratory testing, especially in centralized teams with good communication.

Hypothesis H2.4

The update of documents is hindered by missing traceability links and limited tool support.

The majority of participants agreed that without the support of traceability links, updating the requirements is a cumbersome manual task: *"One requirement is usually spread over many user stories. Then, if requirements change, we need to go through all user stories and figure out which of them cover that requirement. Moreover, every user story is covered by one or sometimes even more tests. Finding the appropriate tests when the requirement is changed is sometimes an art"* (P10).

Although modern management tools like Jira, TFS or Confluence offer features for creating traceability links between artifacts, these features

are not used in more than half of the companies. In one company, the interviewees were even not aware of the existence of such features. Other participants who knew about them would still not use them due to the lack of time: *"We know about the option, but sometimes documents are not linked because of a lack of time. It depends on how much time the consultant has"* (P19).

These tools provide limited or even non-existent support for change propagation and updates. For example, in Jira and Confluence, there is a change notification option. In more than half of the companies that use these tools, the notification option is not used, but practitioners rather choose to rely on communication to find out that some documentation artifacts need to be updated: *"Even if we get an email automatically when some artifacts change, we can see in that email only what has been changed and not why, so we do not know what to do with that change without additional communication. Also, we do not know what else is impacted. We anyway need to check the changed artifact, to see which artifacts are impacted and to manually search one by one in order to change them. This is why the email option does not help much and we decided not to use it, because it will only spam us"* (P20).

Only one participant mentioned that considering the traceability links is unimportant and unnecessary: *"We have small projects and small teams. Everything is based on communication, so we discuss everything in our daily meetings. This is how we know which tests cover which requirements and we do not need to spend additional time to link them [requirements and tests] in tools"* (P17).

Participants stated that a perfect tool for handling requirements and test documentation "keeps all documents at the same place" (P4), "is able to automatically detect impacted test cases when requirements change" (P4, P7), "has a notification system providing a decent level of information" (P20, P6), "has an integrated chat to support better communication" (P4), and "is able to automatically generate test cases from code and requirements" (P3, P7, P4).

Hypothesis H2.5

Organizational culture has a strong impact on the quality and quantity of documentation.

In the previous paragraphs, we have discussed factors that influence the quality and quantity of documentation in companies, such as the distribution of the teams and the software development model. However, we also found that in one company (C8), organizational culture has a stronger impact on the quality and quantity of documentation than the other factors discussed above.

In C8, a product-oriented, agile company, the amount of documentation is much higher than in the other companies. The requirements are first specified in the form of text documents that contain descriptions of the features. These descriptions are then transformed into user stories with defined acceptance criteria. The descriptions and the user stories are then used to write acceptance tests. In this company, all the documents are linked to each other and stored in their document management tool set. If the requirements change, then the original

user stories are kept in the backlog and new user stories are created and linked to the original or previous version. The process is part of the organizational culture of the company and is therefore followed by the employees: *"This is how we worked from the beginning"* (P12), *"I joined this company three months ago. The first thing they showed me is this book [the book written by CEO of the company, which contains a set of process patterns that worked for them in the past, with an aim to make all workers familiar with those patterns]. Then you just follow the good practices and you realize quickly that it works"* (P11).

2.4 Discussion

The main findings of this study are summarized in Table 2.4. In this section, we relate our findings to each other as well as to similar earlier studies.

2.4.1 Engineering could benefit from involving testers.

Several researchers and practitioners (e.g., [Gra02][GW89][Adz09]) have stressed the importance of bringing requirements and testing activities closer together and suggested that this would be beneficial for both sides. Despite the importance of such a practice, it seems not to be widely applied. In fact, for the majority of the interviewed companies, writing the requirements and writing the acceptance tests are separate activities performed by different people. The roles of the people involved in writing the requirements and the tests are summarized in Figure 2.6.

Table 2.4: Summary of the Key Findings

1	**Requirements and tests writing practices - RQ1**
H1.1	Technical people are not involved in the requirements engineering activities.
H1.2	The format of requirements specification depends on the software process applied.
H1.3	There is a strong linear correlation between the number of requirements and tests.
H1.4	Test engineers write acceptance tests based on the requirements that are not necessarily complete.
H1.5	Acceptance tests are mostly written manually, using the typical format.
H1.6	Mixed and centralized teams produce less documentation than distributed ones.
2	**Requirements and tests updating practices - RQ2**
H2.1	Changes of requirements are mainly triggered by the inability to implement the specified requirements.
H2.2	Existing requirements are updated in waterfall-style projects while new user stories are created in agile ones.
H2.3	Acceptance test documentation is less frequently updated in teams with good communication.
H2.4	The update of documents is hindered by missing traceability links and limited tool support.
H2.5	Organizational culture has a strong impact on the quality and quantity of documentation.

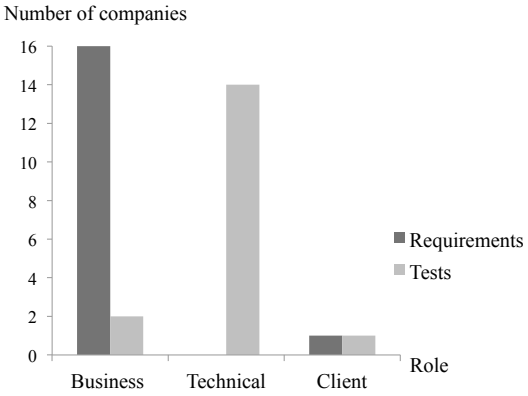


Figure 2.6: Responsible roles for writing the requirements and acceptance test documentation in the companies

Not only testers are not involved, but also other internal stakeholders with technical background, such as developers, do not participate in the requirements engineering activities (H1.1). We found that this results in two major problems. First, the complexity of the requirements is underestimated, which then results in a need to change the requirements later (H2.1). Second, the specified requirements are frequently incomplete, which hinders the writing of tests (H1.4). Similar challenges have also been reported by [LB14][BRB+14][UKKD08]. Uusitalo et al. [UKKD08] identified the unavailability of the testers as one of the reasons for not involving testers in the requirements process. In addition to that, we also identified that the strict separation of roles and the imposed hierarchical structure negatively impacts the practice of including testers in the requirements engineering process.

2.4.2 There is a linear correlation between the number of acceptance tests and the number of requirements in all companies – tests have not replaced requirements.

In an attempt to further consolidate the requirements and testing activities, some agile advocates suggest specifying the requirements in the form of acceptance tests [MM07][Pug10]. When evaluating this practice with a series of experiments, Ricca et al. [RTD+09] found it to increase the understanding of the requirements among different roles. We found that this practice is not applied by any of the companies we interviewed. Although many companies are using acceptance test-driven development (H1.3), the acceptance tests are created based on the requirements (H1.4) and do not replace them (H1.3). The limited use of acceptance tests as requirements could be explained by the non-technical background of the stakeholders who perform the requirements activities. This relates to the results from the study of Bjarnson et al. [BUEB15] who found that customer involvement was one of the challenges that a company faced when using tests as requirements.

We were surprised to find that in the companies we studied, the ratio between the number of requirements and acceptance tests lies between 1:1 and 1:3.3 only, with an average of 1:1.27 (Fig. 2.5). This indicates that acceptance testing is not done properly, since a requirement cannot be fully covered by only one test case. Furthermore, this implies that a similar amount of effort is invested in acceptance testing and in requirements engineering.

2.4.3 The vision document is missing.

In theory, agile approaches propose to create a vision document that (1) represents the strategic intent of product that is built, (2) contains a description of the features to be implemented, (3) clearly defines the problem to be solved and the stakeholders who will benefit from the developed solution, and (4) provides a description of the software in terms of performance, platforms supported, scalability and reliability [Lef10].

In practice, we found that such a document does not exist and even when a similar document exists, it is not kept up-to-date when the requirements change (H1.2). This indicates that in agile development, the focus is on the current goals, while the big picture of the product is missing.

2.4.4 User stories are not sufficient to support tasks dependent on the requirements specification.

With the spread of agile processes, the format used to specify requirements has evolved (H1.2). In agile companies we interviewed, user stories are now the standard way to document requirements. Although user stories have a predefined form, we found evidence that their content is not complete and detailed enough to support the development and testing tasks. This is partially due to not involving technical people in the requirements phase (H1.1). Incomplete requirements hinder the testing process (H1.4) and result in delays. A similar problem has been

reported by previous studies who found that requirements documents and test plans do not always provide enough information to perform the testing accurately and thoroughly [UKKD08] and that, due to incomplete requirements, *"testers need to guess and make up the missing information"* [BRB+14]. Of course, the quality and usefulness of the requirements does not only depend on their completeness, but also on their clarity and level of abstraction as reported by [BRB+14].

2.4.5 Agile development not only changed the format of requirements; it also brought new documentation update practices.

A survey study about the use of documentation reported that requirements documents are mostly not kept up-to-date while tests are usually updated [LSF03]. Things seem to have changed over time. In fact, the agile practices did not only result in changing the requirements format (H1.2) but also changed the update practices for requirements and tests. For instance, we found that user stories are not updated when the requirements change, unless the change is minor. Instead of updating stories, practitioners create new ones (H2.2). Creating new user stories when changes occur, results in having requirements that are up-to-date. However, as old user stories are usually kept, the new requirements are then combined with obsolete ones. In some cases the old user stories are deleted, which means that the history of the change is lost.

The management of tests is different though as they stay outdated in some cases (H2.4). One of the reasons for not updating tests is to the

use of exploratory-based testing and the reliance on informal communication. A few companies mentioned creating new tests instead of updating the existing ones in case the change is non-trivial. This could, however, be costlier than updating the old scripts as shown by [AFK16] [HH08].

2.4.6 Traceability links between documents are not necessarily created.

Traceability links between requirements and test documentation support several activities in software development, testing and evolution [UKKD08] [KVKU09]. Nevertheless, such links are rarely created. Bjarnason et al. [BRB+14] report that traces between requirements and tests can be lacking (even in thought), although they are considered by practitioners as the most basic kind of traceability. This finding is confirmed in our study, as the companies we interviewed also did not have traceability links between requirements and tests. We also found that the lack of traces is an important factor that negatively influences the documentation update practice (H2.4). We identified two main reasons for missing traceability: the lack of time and the limited tool support. Although current tools provide functionalities for trace management, the creation of the links is still a manual task, which requires time and effort. Tools that researchers developed for generating candidate traces seem not to be used in practice yet.

2.4.7 Communication over documentation.

Documentation is in many cases not sufficient to support the software development activities. For example, incomplete requirements (H1.4) force the developers and the testers to recurrently communicate with the requirements engineers in order to specify the tests. Communication is also needed when the documentation is not kept up-to-date (H2.4).

We also found that the quality of the communication impacts the documentation practices in companies. In fact, teams that are mixed and centralized usually produce less documentation than distributed ones (H1.6).

Furthermore, test documentation seems to be less frequently updated in teams with good communication (H2.3). However, although these factors play an important role in the documentation practice, their influence does not overrule the companies' established procedures (H2.5). In fact, we found that well established procedures and habits in terms of documentation management have the greatest influence on the amount and quality of documentation within a company. In project-oriented companies, clients dictate the amount of documentation and the level of details within documents (H1.2), but our results showed that product-oriented companies can also produce high-quality documentation if the right procedures are established. Interestingly, this is the only difference we found between the practices in product-oriented vs. project-oriented companies.

2.5 Related Work

Over the past two decades, many researchers tried to assess the state of practice in the industry, regarding the management of software documentation. Singer [Sin98] and Lethbridge et al. [LSF03] investigated how software engineers use software documentation. Our study covers similar aspects, but uses a qualitative method that allows gaining a more in-depth knowledge on the documentation practice.

Several studies (e.g. [MNA05][NZF04][CR08]) have analysed the management of requirements change. These studies analysed changes in terms of their cause, type, implementation and verification. The results of our study, regarding the causes for a change in requirements, are different and surprising as we found that changes are mostly triggered by the inability to implement the specified requirements.

Stettina and Heijsteck [SH11] studied the time and effort needed for software documentation management in Scrum development teams. In our study, we also investigate documentation management in agile projects, but with a different focus, which is identifying the factors that influence the documentation management process.

Several researchers (e.g. [UKKD08][MM07][MM08][KVKU09]) studied the alignment between requirements engineering and testing. In this work, we also explored the alignment of requirements and tests during software evolution, with the focus on the link between requirements and acceptance test documentation.

Although the question of "how much documentation is enough", has drawn the attention of researchers since more than ten years [Bri03],

the knowledge in this field is still limited. Our study expands this knowledge by providing insight into the documentation practices for requirements and acceptance tests in a sample of companies.

2.6 Conclusion and Future Work

This paper reports the results of an exploratory study about the handling of requirements and acceptance test documentation in seventeen business units from fifteen companies. Our study shows that the processes for managing documentation have changed over the past years. Therefore, new challenges are faced, which requires further research. In this section, we discuss three of these challenges.

First, we found that writing requirements and acceptance tests are performed as two separate tasks by different people. This could result in underestimating the complexity of the requirements and also having incomplete specifications that are not sufficient for performing the testing. Therefore, practitioners need to rely much on oral or informal communication. When communication is hindered or when misunderstandings occur, it often happens that the project gets delayed. Exploring ways to bridge this communication gap is an interesting research area.

Second, in agile companies, the requirements are usually documented in the form of user stories only, while a vision document, which provides a long-term perspective, is usually missing. A deeper understanding of the risks and challenges caused by this practice is worth exploring in the future.

Third, modern documentation management tools do not provide enough support for documentation update. Although they allow tracing artifacts to each other, linking documents is still a manual task that is usually not done due to time constraints. The lack of links hinders analyzing and propagating changes among artifacts. Our study also shows that keeping the history of a change is widely seen as an unnecessary or secondary task, although it improves the quality of requirements documentation and could be used in retrospective analysis, in order to prevent repetition of mistakes.

Chapter 3

Keeping Evolving Requirements and Acceptance Tests Aligned with Automatically Generated Guidance

Original publication:
Keeping Evolving Requirements and Acceptance Tests Aligned with Automatically Generated Guidance
S. Hotomski, E. Ben Charrada, and M. Glinz
24^{th} International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'18), pages 247–264, Springer, 2018 ([HBCG18])

Abstract

[Context and motivation] *When a software-based system evolves, its requirements continuously change. This affects the acceptance tests, which must be adapted accordingly in order to maintain the quality*

of the evolving system. ***[Question/problem]*** *In practice, requirements and acceptance test documents are not always aligned with each other, nor with the actual system behavior. Such inconsistencies may introduce software quality problems, unintended costs and project delays.* ***[Principal ideas/results]*** *To keep evolving requirements and their associated acceptance tests aligned, we are developing an approach called GuideGen that automatically generates guidance in natural language on how to modify impacted acceptance tests when a requirement is changed. We evaluated GuideGen using real-world data from three companies. For 262 non-trivial changes of requirements, we generated guidance on how to change the affected acceptance tests and evaluated the quality of this guidance with seven experts. The correctness of the guidance produced by our approach ranged between 67 and 89 percent of all changes for the three evaluated data sets. We further found that our approach performed better for agile requirements than for traditional ones.* ***[Contribution]*** *Our approach facilitates the alignment of acceptance tests with the actual requirements and also improves the communication between requirements engineers and testers.*

3.1 Introduction

When developing or evolving systems, requirements constantly change and, in most cases, these changes affect other documentation artifacts. In practice, however, impacted artifacts too often are not kept aligned with changing requirements. To a significant extent, this is due to the additional effort required and to insufficient communication of requirement changes [BRB+14] [HBCG16]. Losing the alignment between requirements and other documentation artifacts increases the risk of discovering mismatches between stakeholders' expectations and the actual software behavior only late, leading to unintended costs, delivery delays and unsatisfied customers. For example, when acceptance tests are not kept aligned with changed requirements, testers will report bugs for actual features that were introduced in a change.

In order to keep software documentation aligned and up-to-date when a system evolves, many researchers try to automatically identify which documents are related to each other and which of them are *impacted* by a change [BGW13], [LMOP12]. However, there is little research about how to actually *update* impacted documents, although it would be beneficial to have guidance about what actions to perform [NdlVS13].

In our work, we contribute an approach for keeping acceptance tests aligned with evolving requirements, called GuideGen. GuideGen automatically generates guidance on how to modify impacted acceptance tests when requirements change. We take advantage of the

fact that requirements and acceptance tests have much in common: both are usually written in natural language and contain information about what the system under development is expected to do: requirements specify what should be implemented [SS97] and acceptance tests validate whether the implementation satisfies the requirements of the stakeholders [MSB11]. Due to this similarity, tracing from requirements to acceptance tests is not difficult. Our approach assumes that traces between every requirement and its associated acceptance test(s) exist. If this is not the case, automated trace generation techniques [LMOP12], [ASG+15a] may be used for establishing such traces.

By analyzing changed sentences and words in a requirement, we derive guidance in form of a set of concrete suggestions about what should be changed in the acceptance test(s) associated with a changed requirement. Our tool also provides an easy way for communicating changes and the generated guidance to all interested parties. GuideGen aims at both reducing the effort for aligning acceptance tests with the actual requirements and improving the communication between requirements engineers and developers/testers.

In a previous paper [HBCG17] we presented the principal ideas of our approach together with some examples and a preliminary evaluation. In this paper we describe our method and the algorithms used in detail, give an overview of the GuideGen tool, and present the results of a thorough evaluation with real-world data.

The paper is organized as follows. In the next section we present our approach and its technical components. We then present our

prototype tool in Section 3.3. Section 3.4 describes our evaluation. We discuss our results in Section 3.5. Related work is discussed in Section 3.6. Section 3.7 concludes the paper with a summary and outlook.

3.2 Our Approach

The goal of GuideGen is to identify all *relevant changes* in requirements that require the associated acceptance tests to be adapted and to *generate guidance* in natural language on how to adapt the acceptance tests based on these changes. An overview of our approach is shown in Figure 3.1.

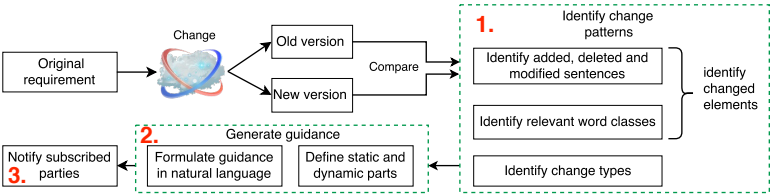


Figure 3.1: Overview of the GuideGen approach

As soon as a requirements engineer applies changes to a requirement and saves them, our approach performs the following steps:

1. Identifying relevant change patterns: by comparing the old and the new version of the changed requirement we identify the elements that have been changed and their change types,

2. Generating guidance: in this step, we formulate suggestions in natural language on how to manage the changes,

3. Notifying subscribed parties: finally, the generated guidance and the changes can be communicated to the interested parties via e-mail.

In the remainder of this section, we present each of these steps in more detail.

3.2.1 Identifying Relevant Change Patterns

The goal of this step is to identify relevant patterns in the changes that are applied to a requirement. A *change pattern* is characterized by the change type (add, delete, or modify) and the changed element (a whole sentence or a word). If the changed element is a whole sentence, the change pattern is "Sentence is added" or "Sentence is deleted". If the changed element is a word, an example of a change pattern is "verb is deleted". *Relevant* change patterns are the ones whose changes require the acceptance tests to be adapted. In particular, relevant change patterns in our approach are the ones that directly or indirectly cause the change of some action, since acceptance tests contain a list of actions to be performed.

To identify the relevant change patterns, we first analyze the changes at a sentence level. Then we proceed by analyzing changes at a word level. Finally we classify each of the detected changes as relevant or

irrelevant.

Analyzing changes at sentence level. In order to identify whether a whole sentence has been added, deleted or modified, we first split the old and the new version of the requirement into sentences using an implementation of the Stanford sentence splitting algorithm [MSB+14]. We get the list of old sentences (oldReq in further text) and the list of new sentences (newReq). Additionally, our tool transforms enumerated sentences into plain sentences. A plain sentence is a sentence without bullet points. An enumerated sentence contains the main part and at least two bullet points, e.g.

"A user can insert: - name,

- surname".

The sentence is transformed into: "A user can insert name" and "A user can insert surname". If a bullet point is added or deleted, the change is treated as an addition or deletion of a plain sentence. For instance, if we add "- e-mail", this change is treated as the addition of the sentence "A user can insert e-mail". Otherwise, the addition of a noun that has no related verbs would be classified as an irrelevant change pattern.

We then compare all the sentences from oldReq with the sentences from newReq by calculating the similarity between them. Based on the similarity, we determine whether the sentence is unchanged, added, deleted or modified. The similarity is calculated using an existing semantic similarity toolkit [RLB+13]. In particular, we use greedy

matching for word to word similarity that is based on WordNet. A flow diagram and the corresponding pseudo code of the algorithm are shown in Figure 3.2.

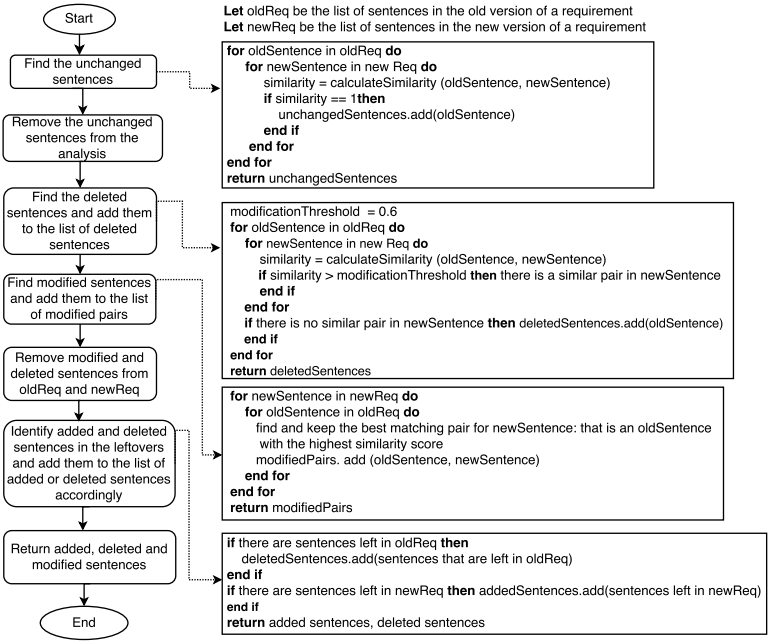


Figure 3.2: The algorithm for identifying added, deleted and modified sentences

If the similarity between a sentence in oldReq and one in newReq is equal to one, that sentence is considered to be unchanged. If a sentence in oldReq does not have a corresponding one in the newReq so that the similarity score between them is greater than the modification

threshold of 0.6[1], then this sentence is deleted. When the similarity score between sentences is above the modification threshold, these sentences are candidates for modified sentences. We choose the best match – a pair of sentences whose similarity score is the highest among other pair candidates. When we remove best matches, unchanged sentences and already identified deleted sentences from the oldReq and the newReq, there might be leftovers. The leftovers in newReq are added sentences and the leftovers in oldReq are deleted sentences. We illustrate this using the following example:

> A user can add new users to the group. ~~The addition of a new user must be first approved by the admin.~~ The admin and the user can modify personal data ~~and the status~~ of that ~~a~~ user. Only user can modify its status. The admin must be logged-in in order to modify personal data of a user.

Added words are green and underlined, removed words are red and struck through, while black words are unchanged.

Figure 3.3 shows the calculated similarities between the old and the new version of the changed requirement.

The first sentence is eliminated from the further analysis because the similarity score is S(1,1) = 1. Since all scores calculated for the second sentence, S(2,2) = 0.36, S(2,3) = 0.11 and S(2,4) = 0.5, are below the modification threshold (0.6), the second sentence in the oldReq is found to be deleted. We defined the modification threshold based on experimentation: we calibrated it to the value that yielded the best results

[1] This is a heuristic value which yielded excellent performance in our evaluation, cf. Sect. 3.4.2.

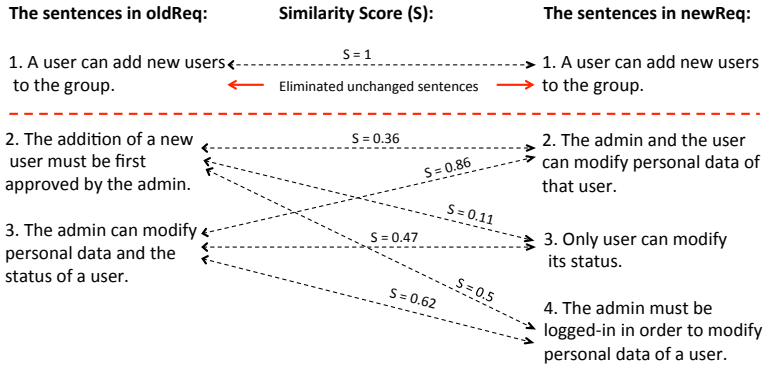


Figure 3.3: Calculated similarity scores for the sentences in the example

for identifying added, deleted and modified sentences. For the third sentence in old-Req we see that there are two matching sentences in the newReq so that the similarity is above the modification threshold: S(3,2) = 0.86 and S(3,4) = 0.62. We choose the best match in this case, i.e., S(3,2). Therefore, the third sentence in the oldReq is modified to the second sentence in the newReq. The third and the fourth sentence in the newReq become leftovers. Since they are both in the newReq we find that these two sentences have been added.

Analyzing changes at word level.

After identifying sentences that have been added, deleted and modified, we proceed to analyze what changes were applied to modified sentences. When a sentence has been modified, we identify word

classes in the sentence and for each of these classes, we identify their change type. For identifying word classes we use Google's implementation of a globally normalized transition-based neural network model, called SyntaxNet [AAW+16]. SyntaxNet determines the word class (e.g., noun, verb) and the grammatical function (e.g., subject, object) for each word in a sentence. SyntaxNet also identifies dependencies between words and represents them with dependency numbers. We use these later when generating guidance (see Sect. 3.2.2). Figure 3.4 shows an example of the output of SyntaxNet.

ID	TEXT	WORD CLASS	DEPENDENCY NUMBER	GRAMMATICAL FUNCTION
1	The	DET	2	det
2	admin	NOUN	4	nsubj
3	can	VERB	4	aux
4	modify	VERB	0	ROOT
5	personal	ADJ	6	amod
6	data	NOUN	4	dobj
7	and	CONJ	6	cc
8	the	DET	9	det
9	status	NOUN	6	conj
10	of	ADP	9	prep
11	a	DET	12	det
12	user	NOUN	10	pobj
13	.	PUNCT	0	punct

ID	TEXT	WORD CLASS	DEPENDENCY NUMBER	GRAMMATICAL FUNCTION
1	The	DET	2	det
2	admin	NOUN	7	nsubj
3	and	CONJ	2	cc
4	the	DET	5	det
5	user	NOUN	2	conj
6	can	VERB	7	aux
7	modify	VERB	0	ROOT
8	personal	ADJ	9	amod
9	data	NOUN	7	dobj
10	of	ADP	9	prep
11	that	DET	12	det
12	user	NOUN	10	pobj
13	.	PUNCT	0	punct

Figure 3.4: The output of SyntaxNet for the old version (left) and the new version (right) of the sentence in the changed requirement

In order to identify whether words have been added, deleted or modified, we adapted the algorithm implemented in a text-based diff engine, called Text_Diff [CH]. Text_Diff detects changes at a phrase level. We process the output from Text_Diff so that we get the changes on a word level.

In the modified sentence from our example: "The admin and the user can modify personal data ~~and the status~~ of that ~~a~~ user", the original

Text_Diff algorithm will detect the addition of the phrases “and the user” and “that” and the deletion of the phrases “and the status” and “a”. We adapted the algorithm so that it detects additions and deletions of each word in these phrases, as presented in Figure 3.5.

The admin <add>and the user</add> can modify personal data ~~and the status~~ of <add>that</add> ~~a~~ user.	The admin <add>and</add> <add>the</add> <add>user</add> can modify personal data ~~and~~ ~~the~~ ~~status~~ of <add>that</add> ~~a~~ user.

Figure 3.5: The original (left) and adapted (right) output of Text_Diff

Classify identified changes into relevant and irrelevant changes.

We consider a change to be relevant if it is likely to impact acceptance tests. Since acceptance tests contain a list of actions to be performed and as actions are generally expressed using verbs in English sentences, we consider verbs as the principal element of analysis in GuideGen. More concretely, we consider a change in a requirement to be relevant if it involves an addition, deletion or modification of a verb or of another word class that relates to a verb such as nouns and adjectives.

If a whole sentence has been added, it is considered to be relevant only if it contains at least one verb. Changes of determiners, adverbs and prepositions are not taken into consideration, since we assume

that they do not influence any actions and, therefore, do not have an impact on acceptance tests.

In our example, the following change patterns are considered to be relevant: (1) deletion of the sentence "The addition of a new user must be first approved by the admin", (2) addition of the noun "user", (3) deletion of the noun "status", (4) addition of the sentence "Only user can modify its status" and (5) addition of the sentence "The admin must be logged-in in order to modify personal data of a user". Only these changes are processed in the next steps.

3.2.2 Generating Guidance

The goal of this step is to generate suggestions about how to modify the affected acceptance tests so that they stay aligned with the changed requirements. An example of a suggestion is *Add new steps or modify existing steps to verify that only user can modify its status.* Every suggestion contains static and dynamic parts.

The static parts of a suggestion differ according to the change patterns identified in the previous step. For instance, if a whole sentence has been added to a requirement, the static part of the suggestion is "Add new steps or modify existing steps to verify that". Accordingly, if a whole sentence has been deleted, the static part of the suggestion is "Delete the steps or their parts which verify that". If a sentence has been modified, the static parts are formulated according to the modification type: whether a verb, subject, object or adjective is added/deleted/modified or a noun is changed from singular to plural,

etc. For instance, if a subject is added, the static parts of the suggestion are "Make sure that now +*{dynamic part}*" and "Add the steps which verify this activity".

The dynamic parts of a suggestion fill the gaps between the static parts. They differ according to the type of the changed element, as shown in Table 3.1. We defined the rules governing the dynamic parts with informal experimentation and by considering typical sentence structures in requirements documents.

If a whole sentence has been added or deleted, the dynamic part contains all words in that sentence. When a changed element is a subject, the dynamic part contains that subject with its determiners and adjectives, the first related verb and all the words that appear after that verb. We use the word index (ID in Figure 3.4) to identify the position of the words. In our example, the following guidance is generated for the added subject "user": "Make sure that now the user can modify personal data of that user."

When the changed element is an object, a verb or an adjective, then the dynamic part contains that element plus its related words. We identify the related words by analyzing word classes, grammatical functions and dependency numbers of words in the modified sentence. Related words for an object are (1) a verb whose index corresponds to the dependency number of the object, (2) a subject whose dependency number refers to the index of the identified related verb and (3) prepositions whose dependency numbers refer to the changed object. We recursively include their related words in the dynamic part. Related words for verbs are (1) directly related subjects, (2) objects, (3) prepositions and (4) adverbs with their related words and corresponding

Table 3.1: Words included in the dynamic part of a suggestion according to the changed element.

Changed element:	Sentence	Noun		Verb	Adjective
		subject/conjunction	object/conjunction		
Words included in the dynamic part:	changed element (all words in that sentence)	changed element with adjectives and determiners, related verb and all words that appear after that verb	changed element, subjects with determiners and adjectives, verbs, prepositions with their objects	changed element, subjects and objects with determiners and adjectives, prepositions with objects, adverbs	changed element, related nouns

indexes and dependency numbers, while related words for adjectives are the nouns that this adjective directly relates to.

If a subject/object is related to another, main subject/object by a conjunction, we identify the words that are related to the main subject/object. In our example, the deleted object "status" has a conjunction to the direct object "data" (see Figure 3.4). Since the verb "modify" with its auxiliary verb "can" is directly related to the object "data", we consider them to be also related to "status". The subject "admin" refers to the verb "modify" and has a related determiner "the", so they are both classified as related words of the deleted object. The preposition "of" directly refers to "status" and it has the related noun "user" with its determiner "a". The determiner "the" is directly related to "status". The words are ordered by the word index and the dynamic part is formulated as *the admin can modify the status of a user.*

Table 3.2 presents the guidance that is generated in our example. The guidance consists of one suggestion per change. Static parts are in boldface, while dynamic parts are italicized.

3.2.3 Notifying Subscribed Parties

In order to ease the communication of changes, we have implemented a notification mechanism that allows requirements engineers to send an automatically generated message to subscribed parties (in particular, testers) when a requirement has been changed. The message contains the summarized changes and the generated guidance. An example is given in Figure 3.7 in the next section.

Table 3.2: The identified relevant change patterns with the corresponding guidance.

Relevant change patterns	Generated guidance
Change 1: deletion of the sentence "The addition of a new user must be first approved by the admin"	**Delete steps or their parts which verify that** *the addition of a new user must be first approved by the admin.*
Change 4: addition of the subject "user"	**Make sure that now** *the user can modify personal data of that user.* **Add the steps which verify this activity.**
Change 7: deletion of the object "status"	**Delete steps or their parts which verify that** *the admin can modify the status of a user.*
Change 10: addition of the sentence "Only user can modify its status."	**Add new steps or modify existing steps to verify that** *only user can modify its status.*
Change 11: addition of the sentence "The admin must be logged-in in order to modify personal data of a user."	**Add new steps or modify existing steps to verify that** *the admin must be logged-in in order to modify personal data of a user.*

If requirements engineers consider a generated suggestion to be irrelevant, they can mark it so that the tool does not include it in the message. For example, if we add a new sentence: "This should be communicated to Tom.", then the generated suggestion "Add new steps or modify existing steps which verify that this should be communicated to Tom." is irrelevant and can be ignored.

3.3 Tool Support

We have implemented our approach in a *prototype tool*, a Java web application. The GuideGen tool allows users to upload the list of requirements from an external Excel file, make changes to each of them and notify subscribers (developers, testers,...) about the changes and the guidance on how to modify the affected acceptance tests.

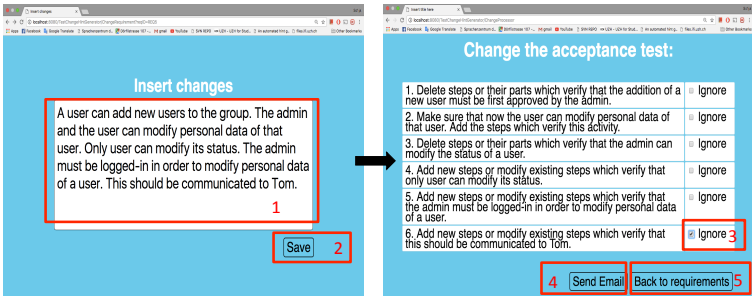
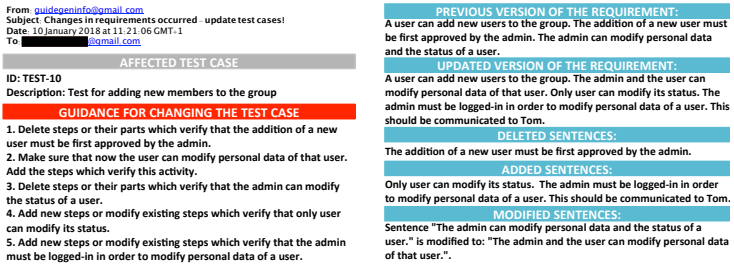


Figure 3.6: User interface (UI) of the tool with highlighted process steps.

Figure 3.6 illustrates the steps taken when using the tool. The left screenshot shows how a user (typically a requirements engineer) can enter changes to a previously selected requirement (step 1) and save them (step 2). Within three seconds, the tool generates guidance consisting of a suggestion for each change and shows it to the user (right screenshot). Suggestions that the user considers to be irrelevant can easily be ignored (step 3). The result can be sent to the subscribed parties in an e-mail generated by the tool (step 4). The user can return to the list of requirements (step 5). Figure 3.7 shows the e-mail generated by the tool for the given example.

3.4 Evaluation

We evaluated GuideGen by applying it to real-world data sets with requirements changes provided by three companies. After pruning the data sets, we ran our tool with the requirements changes contained in


From: guidegeninfo@gmail.com
Subject: Changes in requirements occurred - update test cases!
Date: 10 January 2018 at 11.21.06 GMT+1
To: [redacted]@gmail.com

AFFECTED TEST CASE

ID: TEST-10
Description: Test for adding new members to the group

GUIDANCE FOR CHANGING THE TEST CASE

1. Delete steps or their parts which verify that the addition of a new user must be first approved by the admin.
2. Make sure that now the user can modify personal data of that user. Add the steps which verify this activity.
3. Delete steps or their parts which verify that the admin can modify the status of a user.
4. Add new steps or modify existing steps which verify that only user can modify its status.
5. Add new steps or modify existing steps which verify that the admin must be logged-in in order to modify personal data of a user.

PREVIOUS VERSION OF THE REQUIREMENT:

A user can add new users to the group. The addition of a new user must be first approved by the admin. The admin can modify personal data and the status of a user.

UPDATED VERSION OF THE REQUIREMENT:

A user can add new users to the group. The admin and the user can modify personal data of that user. Only user can modify its status. The admin must be logged-in in order to modify personal data of a user. This should be communicated to Tom.

DELETED SENTENCES:

The addition of a new user must be first approved by the admin.

ADDED SENTENCES:

Only user can modify its status. The admin must be logged-in in order to modify personal data of a user. This should be communicated to Tom.

MODIFIED SENTENCES:

Sentence "The admin can modify personal data and the status of a user." is modified to: "The admin and the user can modify personal data of that user.".

Figure 3.7: The e-mail message generated for the example given in Fig. 3.6.

the data sets and generated guidance for how to change the associated acceptance tests. The quality of the generated guidance was then assessed by experts from the three companies.

3.4.1 Study Design

Data collection and analysis. We obtained data sets containing information about changes of requirements from three companies (Table 3.3). For our evaluation, we needed data records containing the old and the changed version of a requirement and the associated acceptance tests. Table 3.4 characterizes the data sets.

We pruned the received data sets as follows: (1) We omitted all requirements that had not been changed at all or did not have acceptance tests associated with them. (2) We removed irrelevant changes such as added or deleted punctuation marks, spaces or empty lines. The

Table 3.3: Characteristics of the companies that provided us data sets from one of their projects.

Comp.	Domain of activity	Software process model	# of employees in total	# of employees on the project	Country
C1	Access control and security solutions	Agile(Scrum)	~16000	~120	Switzerland
C2	IT integration, cloud services	Agile (Scrum)	~500	~100	Serbia/ Germany
C3	Automation for warehouses and distribution centers	Waterfall	~2500	~500	Switzerland

Table 3.4: Characteristics of the data sets used in our evaluation study.

Company/ Data set	Type of requirements	# of requirements in the data set	# of considered requirements	# of evaluated changes
C1/DS1	User story	157	20	28
C2/DS2	User story	30	30	37
C3/DS3	Classic textual requirement	5301	398	197

pruning yielded a total of 448 changed requirements. Our tool filters out semantically irrelevant changes such as addition or deletion of determiners or corrections of typos. On the other hand, for several requirements there was more than one change. So we eventually could evaluate a total of 262 changes (28 for C1, 37 for C2 and 197 for C3).

Running the tool. For every of the 262 evaluated changes, we generated guidance for how to change the associated acceptance tests

using our tool prototype. We uploaded the old version of the requirements into the tool, replaced each of them with the new version, and recorded the generated guidance.

Assessing the quality of the generated guidance. The generated guidance was assessed by experts from the three companies. An overview of the experts and their experience is provided in Table 3.5.

Table 3.5: Characteristics of the experts who participated in the study.

Company	Participant	The role of participant	Years of experience in IT	Years on the current position
C1	P1	Requirements engineer	10	4
C1	P2	Senior test analyst	12	4
C2	P3	Requirements engineer	6	3
C2	P4	Senior test engineer	7	4
C3	P5	Requirements engineer	10	5
C3	P6	QA manager	12	6
C3	P7	Test engineer	4	4

95 changes were fully assessed by two or three experts. We created a questionnaire[2] in which, for every requirement, we presented the old and the changed requirement, the associated acceptance tests and the guidance for changing the acceptance tests generated by our tool. For each suggestion provided in the guidance, we asked six questions to assess the quality of the suggestion: (1) Is the suggestion correct in terms of actions that need to be performed? (2) Is it grammatically correct? (3) Is it complete? (4) Does the expert understand what has been suggested by the tool? (5) Would the expert be able to perform

[2] https://data.mendeley.com/datasets/cs6ymc7tnj/1
For confidentiality reasons, the file does not contain the real data from our data sets, but only the example shown in this paper.

an update of the impacted acceptance test without any further clarifications? (6) Is the suggestion redundant or unnecessary? Finally, we asked whether there is anything missing from the guidance for a changed requirement (i.e., from the set of all suggestions generated for that requirement). Questions 1-3 and 5 had to be answered on a five-point Likert scale (from "strongly disagree" to "strongly agree"). In case of non-agreement, the expert was asked to provide an explaining text. Question 4 was a yes/no question, while Question 6 and the final question about missing suggestions were answered as free text.

In company C3, due to limited availability of the experts, only 30 suggestions could be thoroughly assessed by all three experts. The suggestions generated for the remaining 167 changes could only be assessed for correctness by a single expert.

When the experts had finished answering the questionnaire for all changed requirements assigned to them, we conducted a short interview where we asked them seven questions about the usefulness and applicability of our approach[3].

3.4.2 Results

In this sub-section we present the results of the assessment of the generated guidance by the experts and some key insights from the follow-up interviews.

[3] https://tinyurl.com/ycl8jyak

All 262 changes were correctly identified in terms of the change type, showing that the algorithm for identifying added, deleted and modified sentences with a modification threshold of 0.6 performs accurately. Table 3.6 presents the results of the evaluation of the guidance generated for 95 changes in requirements by the experts.

For calculating the percentages in Table 3.6 for the questions answered on a Likert scale, we interpreted the values 4 ("Agree") and 5 ("Strongly agree") as "yes". Analogously, we interpreted 1 ("Strongly disagree") and 2 ("Disagree") as "no". 3 ("Neutral") was interpreted according to the textual explanation provided by the experts. From eleven such answers three were interpreted as "yes" and eight as "no".

Table 3.6 shows that in C1 and C2 the experts assessed more than 80% of the suggestions as correct in terms of actions. In C3 one expert was more negative than the other two, especially regarding the correctness in terms of actions. This is due to a misunderstanding: expert P5 classified all redundant suggestions as wrong in terms of actions, i.e., when they were actually correct, but unnecessary. So we can consider the correctness of our guidance for data set 3 to be at least 66.7%.

Fig. 3.8 shows a change (in the acceptance criteria of a user story) where GuideGen does not work such well. According to the experts, the text means that Section 3 of a web page contains a label "Doctors' corner" and a button "Register your practice". When a user clicks on the button, a pop-up window is displayed. The change in the requirement is that an additional message shall be displayed in this window.

Table 3.6: The quality of the generated suggestions based on an assessment by industrial experts.

Generated in total/ assessed	Company/ Participant	Correct in terms of actions	Grammatically correct	Complete	Understa-ndable	Self-expla-natory	Redundant/ unnecessary	Missed changes
28/28	C1/P1	89.2%	82.1%	100%	100%	75%	7.1%	3.6%
	C1/P2	89.2%	82.1%	100%	100%	75%	7.1%	3.6%
37/37	C2/P3	81%	67.5%	94.6%	100%	75.6%	10%	5.4%
	C2/P4	81%	67.5%	94.6%	100%	75.6%	10%	5.4%
197/30	C3/P5	50%	86.6%	96.6%	93.3%	70%	50%	3.3%
	C3/P6	70%	80%	93.3%	100%	73.3%	30%	3.3%
	C3/P7	66.7%	86.6%	96.6%	93.3%	73.3%	33.3%	3.3%

"- The section 3 contains:
-- Doctors' corner
-- Register your practice opens a form inline or a popup with:
--- Name of your practice (mandatory)
--- Contact phone (mandatory)
--- Contact e-mail (mandatory)
--- Give us your contact details and we will get back to you soon!"

Figure 3.8: Example of a changed requirement from C2. Added text is in green and underlined.

For this change, the GuideGen tool generated the following suggestion, which the experts considered to be wrong both in terms of actions and grammatically: "Add new steps or modify existing steps which verify that the section 3 contains register your practice opens a form inline or a pop-up with give us your contact details and we will get back to you soon!". This result may indicate that our approach does not perform well on ill-structured texts (the experts confirmed that this text is not formulated well). However, it may also indicate that our treatment of enumerations (cf. sentence level analysis in Sect. 3.2.1) needs improvement.

The last column in Table 3.6 presents the number of changes that were relevant, but not detected by GuideGen. In C1 a noun without any related verbs was added. This was classified as an irrelevant change and hence no guidance was generated. Further, the current version of our tool does not consider the change of numerical values as a relevant change pattern. Hence, no guidance was generated for two such cases in C2 and one in C3. This problem will be fixed in the next release of the tool.

As stated above, the guidance for 167 changes in requirements from company C3 could not be evaluated fully due to limited availability of the experts. Table 3.7 shows the results of the assessment of the generated suggestions for these changes.

Table 3.7: Suggestions assessed for correctness in terms of actions by a single expert only.

Company/ Participant (role)	Assessed suggestions	Correct in terms of actions	Wrong (rephrasing only)	Wrong (only clarifications or notes added or deleted)	Wrong (due to tool limitations)
C3/P6(QA)	167	70.6%	10.2%	13.8%	5.4%

We found that 70.6% were correct in terms of actions, while 24% were incorrect because the changes only rephrased a requirement or added or deleted only clarifications or notes. A small percentage (5.4%) of wrong suggestions were due to limitations of our prototype tool (e.g., wrongly identified dependencies).

Next, we present the main findings from the follow-up interviews with the experts regarding the overall usability and usefulness of GuideGen. All experts stated that GuideGen can be helpful in communicating changes on time and with less effort, it can help test engineers to make a decision on how to update acceptance tests and they would be willing to slightly adapt their style of writing requirements in order to ensure better quality of guidance. Four experts emphasized that one of the reasons for wrongly generated guidance was the poor quality of the requirements. They stated that suggestions can be too general, but that

this is directly related to the level of detail specified in the requirements. The experts from C1 stated that the approach would be even more useful if it could highlight the parts of the acceptance tests that should be changed directly in the acceptance test document. With respect to the usability of the tool, P1 and P2 suggested an improvement of the user interface so that the tool navigates directly to the steps that are suggested to be changed.

3.4.3 Threats to Validity

Internal and construct validity. Our evaluation strongly depends on the expertise of the people who assessed the guidance generated by GuideGen. In order to foster validity, we aimed at assessing each guidance by at least two experts. In company C3, due to limited availability of experts, we could assess only 30 cases this way, while the rest was evaluated only in terms of correctness by a single expert. We tried to mitigate this problem by including all types of changes in the fully evaluated sample from company C3. Even with this restriction, the workload for the experts was high, since they needed to answer six questions per 28 and more suggestions, which might impact the quality of their answers. Therefore, we provided an online access to the questionnaire, so that the experts could answer the questions in iterations.

External validity. The generalizability of our results is limited by the fact that our evaluation covers data sets from only three companies. We tried to improve generalizability by including both agile and traditional requirements artifacts as well as different types of changes in

our data sets. Although the study involves only seven participants, we had at least two participants per data set and we tried to keep diversity in terms of roles, so that requirements engineers and test managers are included.

3.5 Discussion

The results presented in Table 3.6 show that the quality of the generated guidance differs from company to company. This is not surprising as the outcome of our natural language processing techniques depends on the type and quality of requirements artifacts and on the content that is being changed in these artifacts.

GuideGen performs better for user stories than for traditional requirements. This is probably due to the fact that user stories typically are more concise and describe features more precisely than traditional requirements do. Further, text changes in traditional requirements documents often do not bring any novelty to the feature that is being described, but only provide clarifications or simply rephrase the text.

The complexity of a sentence also affects the quality of the guidance generated. On the one hand, very short or incomplete sentences affect both the correctness and completeness of suggestions and may even cause the omission of relevant changes. On the other hand, long, complex sentences which contain one or more relative clauses or statements in parentheses may cause problems: word classes, their

grammatical functions and dependencies between words in a sentence may be wrongly identified, which leads to wrongly generated guidance.

Our approach currently cannot recognize certain types of irrelevant changes, for example, when mere comments such as "This should be communicated to Tom" are added. Wrong suggestions are generated in this case. However, our tool allows a requirements engineer to remove such false positives easily before communicating changes and generated guidance to subscribers (cf. Fig. 3.7).

GuideGen needs only sets of old and changed requirements (and their associated acceptance tests) as input. This is both a strength and a limitation. It is a *strength* because with our tool, requirements engineers can easily communicate requirements changes together with guidance on how to change the acceptance tests that correspond to the changed requirements. On the other hand, it is a *limitation*, as our tool does not analyze which artifacts are impacted by a changed requirement. This problem is addressed by research on automated traceability and change impact analysis [ASG+15a], [LMOP12], [MMS05].

3.6 Related Work

Many researchers investigate requirements traceability for supporting change impact analysis. For example, Antoniol et al. [ACC+02], Marcus and Maletic [MMS05], De Lucia et al. [LMOP12] and Hayes et al. [HDS06] use information retrieval methods to ensure automated

traceability for change impact analyses. Others employ natural language processing. For example, Arora et al. [ASG+15a] analyze the impact of changes in a requirement on other requirements in a system using NLP methods. However, all these approaches focus on identifying which requirements or other artifacts are impacted by a change in a requirement, while we investigate how to manage the change and which actions to perform in order to keep requirements and acceptance tests aligned.

Bridging the communication gap among people involved in developing a system draws attention of researchers and practitioners. Sinha et al. [SSC06] define and explain the communication problems when managing requirements in a distributed environment. Bjarnason and Sharp [BS17] and Adzic [Adz09] emphasize the communication problems between requirements engineers, developers and testers in agile projects. By generating guidance in natural language that can be easily communicated to the interested parties via e-mail, our approach supports easy and timely communication of changes between requirements engineers and developers/testers.

3.7 Conclusions

Summary. We presented GuideGen, a tool-supported method for automatically generating guidance on how to align acceptance tests with evolving requirements. With a correctness score of more than 80% for real-world agile requirements and around 67% for traditional requirements, our approach provides useful guidance for maintaining

acceptance tests and keeping them aligned with the evolving requirements.

Future Work. We will improve GuideGen based on the evaluation results and then perform a more thorough evaluation of its overall usefulness and usability.

Chapter 4

GuideGen – A Tool for Keeping Requirements and Acceptance Tests Aligned

Original publication:
GuideGen – A Tool for Keeping Requirements and Acceptance Tests Aligned
S. Hotomski and M. Glinz
40th International Conference on Software Engineering 2018 (ICSE'18), pages 49–52, ACM, 2018 ([HG18a])

Abstract

When changes in requirements occur, their associated tests must be adapted accordingly in order to maintain the quality of the evolving system. In practice, inconsistencies in requirements and acceptance tests—together with poor communication of changes—lead to software

quality problems, unintended costs and project delays. We are developing GuideGen, a tool that helps requirements engineers, testers and other involved parties keep requirements and acceptance tests aligned. When requirements change, GuideGen analyzes the changes, automatically generates guidance on how to adapt the affected acceptance tests, and sends this information to subscribed parties. GuideGen also flags all non-aligned acceptance tests, thus keeping stakeholders aware of mismatches between requirements and acceptance tests. We evaluated GuideGen with data from three companies. For 262 non-trivial changes of requirements, the suggestions generated by GuideGen were correct in more than 80 percent of the cases for agile requirements and about 67 percent for traditional ones.

Demo video: https://vimeo.com/254865530

4.1 Introduction

Weak alignment of requirements-related activities with validation and verification tasks leads to software quality problems, unintended costs, wasted effort and delays. In particular, when requirements documents are not aligned with their corresponding acceptance tests, features might be incorrectly verified or not verified at all [BRB+14]. Maintaining the alignment is a difficult task due to many factors, such as poor communication of changes among team members and manual tracing between requirements and acceptance tests. For instance, when a requirements engineer changes a requirement, but does not timely communicate this to developers and testers, testers will report bugs for newly added or removed features. Time and effort is wasted for understanding the causes of such problems and making the documentation consistent again [HBCG16]. Furthermore, communicating changes explicitly is time-consuming and prone to misunderstandings. Problems also stem from using different tools for managing requirements and acceptance tests where traces between artifacts must be established and maintained manually. In some companies, requirements and tests are still captured in spreadsheets or text documents, having barely or no traces between documents.

We are developing GuideGen, a tool that keeps requirements and acceptance tests closely together and aligned when requirements evolve. The core idea of our tool, which distinguishes it from other document management tools, is that it automatically generates guidance on how to adapt the impacted acceptance tests according to changes in requirements. In such a way, GuideGen supports requirements

engineers in easily communicating requirements changes to testers and test engineers in properly adapting acceptance tests when their related requirements change. In addition, GuideGen automatically creates the traces between requirements and acceptance tests as soon as a new acceptance test is added for a requirement.

In this paper, we focus on the main features of the tool and how they are used. The technical details as well as the evaluation of the GuideGen approach are described in [HBCG17] and [HBCG18].

We describe the main features of the tool in Section 4.2 and summarize the evaluation in Section 4.3. Section 4.4 briefly discusses related work.

4.2 The GuideGen Tool

GuideGen is a web application, written in Java using Servlet and JSP technology [Hal03]. It is deployed on Apache Tomcat [WGC+04]. On the one hand, GuideGen supports requirements engineers in maintaining the requirements of a system and in communicating all changes of requirements to testers, developers and other interested parties on-time and with almost no effort. On the other hand, GuideGen supports testers, who maintain acceptance test documents, by providing them with guidance on how to modify impacted tests so that they stay aligned with the modified requirements. In addition, by flagging all non-aligned acceptance tests, any stakeholder can easily see which acceptance tests are currently mis-aligned with their corresponding

requirement—be it that tests do not exist yet or that they have not been updated after changes in the requirements.

The current version of GuideGen is limited to one acceptance test per requirement. This is no severe limitation: in a previous study [HBCG16] we found that a one-to one correspondence between requirements and acceptance tests frequently occurs in practice.

4.2.1 Using GuideGen

Upon starting the application, a user can (1) upload a collection of existing requirements and acceptance tests into GuideGen, (2) view the current list of requirements and acceptance tests, or (3) add a new requirement. The resulting list is shown in Figure 4.1.

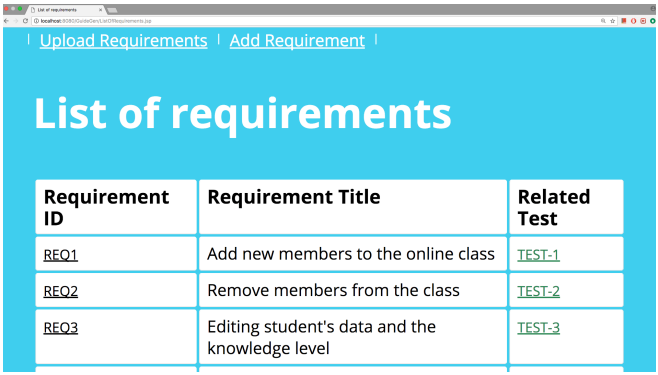


Figure 4.1: List of requirements and their acceptance tests.

The acceptance tests shown in Figure 4.1 are all aligned with their requirements—they are green and no warning flags are shown. When

a user clicks on the identifier of a requirement or acceptance test, GuideGen shows the selected requirement or test in detail and the user can edit it. Subsequently, we describe three typical scenarios for working with GuideGen.

Scenario 1: Changing a requirement. Let's assume that a requirements engineer selects and edits the requirement REQ3:

> A student can edit his personal data ~~and the knowledge level.~~ Only teachers can edit students' knowledge level.
> - The knowledge level can be: – beginner, – intermediate, – advanced, – proficient. This change is due to users' feedback.

Added words are green and underlined, removed ones are red and struck through, while unchanged text is shown in black. As soon as the requirements engineer saves the changes, GuideGen identifies the relevant change patterns and, within about three seconds, generates guidance on how to adapt the associated acceptance test. Figure 4.2 shows the guidance generated for the previously presented changes.

The guidance generated by GuideGen consists of a suggestion per added, deleted or modified sentence. GuideGen lets the requirements engineer review the suggestions and allows her to mark those that she considers irrelevant or wrong (Fig. 4.2). As soon as she confirms, GuideGen automatically creates an e-mail and sends it to all GuideGen users who have subscribed for receiving change alerts. This e-mail consists of three parts: (1) the test case to be modified, (2) the generated suggestions, and (3) the original and the changed requirement.

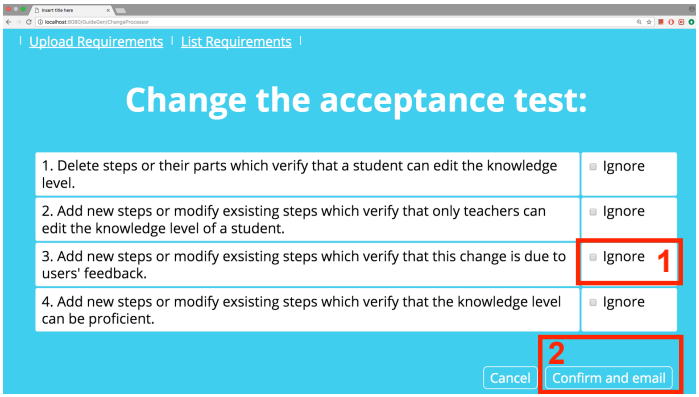


Figure 4.2: List of generated suggestions. A user may ignore suggestions (1) and e-mail the other ones to interested parties (2).

GuideGen also flags non-aligned tests in the list of requirements and acceptance tests with yellow color and a warning sign, as shown in Figure 4.3.

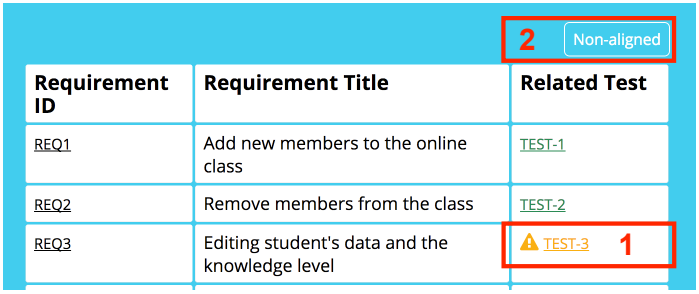


Figure 4.3: The warning flag showing a mismatch (1) and the button for displaying the non-aligned documents only (2).

Scenario 2: Adapting an acceptance test. Let's assume that a tester receives an e-mail about the change in requirement REQ3 as described above, reads the message, and decides to adapt the impacted acceptance test. After launching GuideGen, the tester clicks TEST-3 which is flagged as non-aligned by GuideGen (see Figure 4.3). GuideGen then displays a form for editing the test case (see Figure 4.4). In the edit field on the left, the current version of the test is displayed. On the right side, the suggestions generated by GuideGen about how to change the test case are displayed. The tester can now edit the test according to the given suggestions, without having to analyze the changed requirement manually. When he saves his changes, GuideGen will remove the warning in the requirements list and display the test-ID in green.

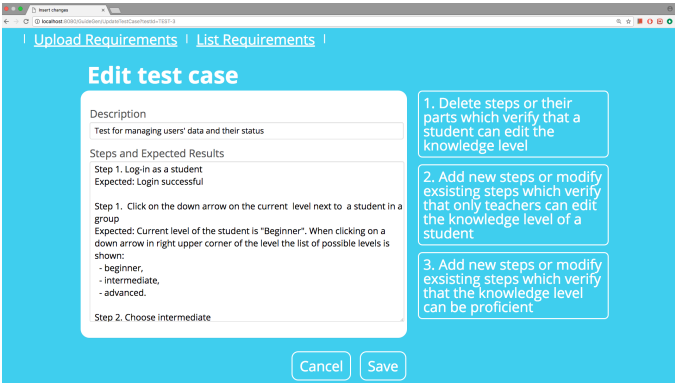


Figure 4.4: The form for editing TEST-3.

Scenario 3: Overview of non-aligned tests. A stakeholder can get an overview of all acceptance tests which are currently not aligned by opening the list of requirements and acceptance tests in GuideGen

(Figure 4.3) and filtering it for non-aligned entries by clicking the "Non-aligned" button.

4.2.2 The Algorithms Used in GuideGen

In this subsection, we briefly describe how our approach works. GuideGen first identifies relevant change patterns and then generates the text of the suggestions. For more details see [HBCG18] or Chapter 3.

Identifying relevant change patterns. Relevant change patterns characterize those changes in requirements that require the associated acceptance test to be adapted. We classify the changes into relevant and irrelevant ones by analyzing changes both on the sentence level and on the level of individual words.

In the analysis on the *sentence level*, we identify added, deleted and modified sentences. First, we split the old version of the changed requirement (oldReq in further text) and the new version (newReq) into sentences using an implementation of the Stanford sentence splitting algorithm [MSB+14]. We then compare the sentences from oldReq with the sentences from newReq by using a semantic similarity toolkit [RLB+13]. If the similarity between a sentence in oldReq and one in newReq is equal to one, that sentence is unchanged. For distinguishing between deleted and modified sentences, we use an experimentally determined similarity threshold of 0.6. If for a sentence s in oldReq there is no sentence in newReq with a similarity score above the similarity threshold, we consider s to be deleted. Otherwise, s has

been modified. We identify the corresponding sentence s' in newReq by choosing the pair (s, s') with the highest similarity score among all candidate pairs.

When we remove best matches, unchanged sentences and already identified deleted sentences, the leftovers in newReq are added sentences and the leftovers in oldReq are deleted sentences.

For modified sentences, we then analyze the changes on the *word level*. First, we split each modified sentence into a list of words. Then our diff engine determines whether the words have been added, deleted, modified or are unchanged. We adapted the algorithm implemented in Text_Diff [CH], so that we get the changes at a word level, instead of a phrase level as in the original implementation. We then identify a word class (e.g. noun, verb) and grammatical function (e.g. subject, object) for each word. For identifying word classes we use SyntaxNet [AAW+16]. SyntaxNet also identifies dependencies between words, presented as dependency numbers, which is used later when we generate suggestions. An example of the output of SyntaxNet is shown in Figure 4.5.

Finally, we classify the changes into relevant and irrelevant ones. As acceptance tests contain a list of actions to be performed, and actions are generally expressed using verbs in English sentences, we consider a change in a requirement to be *relevant* if it involves an addition, deletion or modification of a verb or of another word class that relates to a verb, such as nouns (subjects, objects). Changes of other word classes, such as determiners, relative pronouns or prepositions are irrelevant, since we assume that they do not influence any actions.

ID	TEXT	WORD CLASS	DEPENDENCY NUMBER	GRAMMATICAL FUNCTION
1	A	DET	2	det
2	student	NOUN	4	nsubj
3	can	VERB	4	aux
4	edit	VERB	0	ROOT
5	personal	ADJ	6	amod
6	data	NOUN	4	dobj
7	and	CONJ	6	cc
8	the	DET	10	det
9	knowledge	NOUN	10	nn
10	level	NOUN	6	conj
11	.	PUNCT	0	ROOT

Figure 4.5: The SyntaxNet output for the old version of the first sentence in the example shown above.

Generating guidance. For all relevant changes, we now formulate a list of suggestions on how to adapt the affected test cases. We identify static parts first, then we identify dynamic parts and finally we combine them to formulate a suggestion.

The static parts are identified by analyzing the change patterns. For instance, when a whole sentence has been added, the static part is "Add new steps or modify existing steps to verify that". When a sentence has been deleted, the static part is "Delete steps or their parts which verify that". If a sentence has been modified, the static parts are formulated according to the modification type: whether a verb, subject, object, adjective or number has been added, deleted or modified. For instance, if a subject has been added, the static parts of the suggestion are "Make sure that now +*{dynamic part}*" and "Add the steps which verify this activity."

We then identify *the dynamic parts*. When a whole sentence has been added or deleted, the dynamic part contains that sentence. When a

sentence has been modified, we formulate the dynamic parts by finding and sorting the related words of the changed word. For instance, when "the knowledge level" has been deleted, we identify that the object "level" refers directly to the noun "data" by a conjunction (see Figure 4.5). In this case we find verbs and subjects that are related to the main object ("data"). The verb "edit" with its auxiliary verb "can" is directly related to the noun "data" and the subject "student" with its determiner "a" directly refers to the verb "edit". The noun "knowledge" with its determiner "the" directly refers to the object "level". We sort the words by their index and formulate the dynamic part of the suggestion: "a student can edit the knowledge level". Since the change pattern is "an object is deleted", the static part is "Delete steps or their parts which verify that". Finally, when we combine the static and the dynamic part, the following suggestion is generated: *"Delete steps or their parts which verify that a student can edit the knowledge level."*

GuideGen is applicable for requirements and acceptance tests written in (free-form or structured) natural language. We have not explored generating suggestions for how to change the test code when using automated, executable tests.

4.3 Evaluation

In this section, we summarize the results of a first evaluation of the GuideGen approach with real-world data. The details are described in [HBCG18] (Chapter 3).

Study design. We obtained 262 requirements changes from three companies. Companies C1 and C2 work with user stories, while company C3 writes traditional textual requirements. For every change, we used GuideGen to generate guidance about how to change the associated acceptance test. We then asked experts from the three companies to evaluate the generated suggestions with respect to seven questions: (1) correct with respect to suggested actions? (2) grammatically correct? (3) complete? (4) understandable? (5) self-explanatory?, (6) unnecessary?, and (7) any missed changes?

Results. For companies C1 and C2, the experts found more than 80% of the suggestions to be correct in terms of actions. For C3, there was some disagreement among the experts resulting in 66.7% of the suggestions being considered correct on average. Grammatical correctness ranged from 67.5% (C2) to over 80% (C1 and C3). Over 93% of all suggestions were considered to be complete and understandable by all experts. Over 70% were considered to be self-explanatory. With respect to necessity, results were good for C1 and C2: the experts found only 7 to 10% of the suggestions to be unnecessary. In C3, however, two experts considered about 30% of the suggestions unnecessary and one expert even found 50% to be unnecessary. The rate of missed changes (i.e., no suggestion was generated although it would have been appropriate to create one) was only 3 to 6%. When comparing our results with a gold standard of 100% correct suggestions, GuideGen achieves a very high recall (94-97%) with rather high accuracy.

In follow-up interviews, all experts deemed GuideGen to be useful for communicating changes on time and with less effort, and helpful for test engineers to update acceptance tests. They also preferred

the explicit guidance produced by GuideGen over just being notified about changes in requirements.

4.4 Related Work

In order to keep software documentation consistent when a system evolves, researchers propose information retrieval [CHBC+07], [LMOP12] or natural language processing [ASG+15a] methods to automatically identify which documents are related to each other and which of them are impacted by a change. However, there is little research about how to actually update impacted documents, although it would be beneficial to have guidance about how to handle the changes and what actions to perform [NdlVS13]. Our tool provides concrete suggestions for testers on how to handle changes.

Many researchers studied the role of communication in the success of software projects and its challenges [SSC06], [BS17], [Adz09]. Some suggest that face-to-face communication should be the main source of knowledge sharing, since writing e-mails or documents is time-consuming [MM04]. However, having only verbal communication introduces the risk of misunderstandings and miscommunicated changes due to many factors, such as geographic, temporal, cultural, and linguistic distance [NBR11]. GuideGen reduces the effort needed for communicating changes by automatically providing e-mails with information regarding the change.

4.5 Conclusion

Summary. We presented GuideGen, a tool that (1) automatically generates guidance on how to align acceptance tests with evolving requirements, (2) provides e-mail notifications with that guidance and a summary of changes, and (3) sets warning flags that make stakeholders aware of mismatches between requirements and associated acceptance tests. The evaluation of our approach shows promising results with respect to the correctness, completeness and understandability of the generated guidance.

Limitations. Currently, GuideGen is a standalone tool, which limits its applicability in projects that use existing tool chains for managing requirements and tests. This is due to the fact that our tool is part of ongoing research where the focus is on principles rather than on features. As mentioned above, the current version is also limited to one acceptance test per requirement. Further, GuideGen does not check whether the generated suggestions are actually followed when an acceptance test is updated.

Future Work. We are currently evaluating the GuideGen approach with respect to its usefulness and applicability in industrial practice. We also plan to address the current limitations of the GuideGen tool.

Chapter 5

Usefulness of GuideGen and its Applicability to Real-World Projects

Original publication:
A Qualitative Study on Using GuideGen to Keep Requirements and Acceptance Tests Aligned
S. Hotomski and M. Glinz
26^{th} IEEE International Requirements Engineering Conference 2018 (RE'18), pages 29–39, IEEE, 2018 ([HG18b])

Abstract

Software requirements constantly change, thus impacting all other artifacts of an evolving system. In order to keep the system in a consistent state, changes in requirements should be documented and applied accordingly to all affected artifacts, including acceptance tests. In practice, however, changes in requirements are not always documented nor

applied to the affected acceptance tests. This is mostly due to poor communication, lack of time or work overload, and eventually leads to project delays, unintended costs and unsatisfied customers. GuideGen is a tool-supported approach for keeping requirements and acceptance tests aligned. When a requirement is changed, GuideGen automatically generates guidance in natural language on how to modify impacted acceptance tests and communicates this information to the concerned parties.

In this paper, we evaluate GuideGen in terms of its perceived usefulness for practitioners and its applicability to real software projects. The evaluation was conducted via interviews with 23 industrial practitioners from ten companies based in Europe. The results indicate that GuideGen is a useful approach that facilitates requirements change management and the communication of changes between requirements and test engineers. The participants also identified potential for improvement, in particular for using GuideGen in large projects.

5.1 Introduction

Changes in software requirements are inevitable and they affect all other software activities, including acceptance testing. Although requirements engineering and acceptance testing are mostly performed separately, there is much synergy between them. Requirements describe what a system should do, while acceptance tests verify that the actual behavior and properties of the system comply with the specified requirements [MSB11]. Therefore, requirements and their corresponding acceptance tests should always be aligned [UKKD08].

In order to achieve and maintain this alignment, all changes in requirements should be documented and communicated on time to the testers so that they can change the acceptance tests accordingly. However, in practice, keeping requirements and acceptance tests aligned and up-to-date is a challenging and expensive task [HBCG16]. Therefore, automation in requirements change management [AHB16], as well as guidelines and tool support [KHO17] are needed.

GuideGen is a tool-supported approach that facilitates keeping requirements and acceptance tests aligned [HBCG18]. When a requirement is changed, GuideGen automatically generates guidance on how to modify the affected acceptance test. We previously evaluated GuideGen in terms of the correctness of the generated guidance [HBCG18]. However, we did not perform a thorough qualitative evaluation of the actual usefulness of GuideGen in practice.

In this paper, we aim at closing this gap by evaluating GuideGen in terms of its usefulness to practitioners and applicability to their

projects, using a qualitative experiment [Kit96]. We presented GuideGen to practitioners using their own data and then conducted thirteen interviews in ten companies with a total of twenty-three experienced practitioners. In order to obtain a baseline for comparison, we started the interviews with an exploratory part where we investigated the current challenges that the participants are faced with when requirements change. Then we assessed the usefulness of GuideGen in practice by asking the participants which of their current challenges can be mitigated by using GuideGen and how applicable and useful GuideGen would be in their current projects.

The results show that the challenges about aligning requirements and acceptance tests found in earlier studies (e.g., [HBCG16], [BRB+14], [LB14]) are still present, indicating that new approaches are needed for managing changes in requirements and keeping them aligned to acceptance tests. The principal idea of GuideGen was rated to be useful or even extremely useful by all interviewed practitioners, and even in the practical setting of the current projects, 20 of the 23 participants found GuideGen to be useful or extremely useful. The participants also identified things to be changed or added to the GuideGen tool in order to make it fully applicable in practice.

The remainder of the paper is structured as follows. We first give a short overview of the GuideGen method and tool in Section 5.2. In Section 5.3, we describe our research methodology. In Section 5.4 we present the results and discuss them in Section 5.5. Related work is discussed in Section 5.6. Section 5.7 concludes the paper.

5.2 GuideGen

GuideGen[1] is a tool-supported approach for automatically generating guidance on how to update acceptance tests when their corresponding requirements change [HBCG18], [HG18a]. To make this paper more self-contained, we briefly describe the GuideGen method and tool with a running example from the healthcare domain. Figure 5.1 shows the list of requirements for a patient management system in the GuideGen tool.

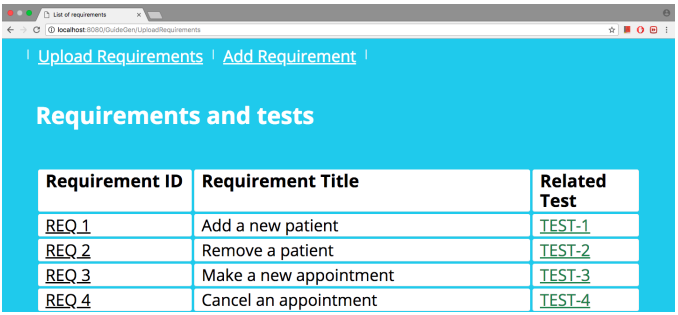


Figure 5.1: A list of requirements and their acceptance tests

We assume that a requirements engineer needs to change requirement REQ4 about cancelling an appointment. She clicks on the requirement ID in the tool and gets a window where she can edit the requirement as follows (added parts are in green and underlined):

[1] http://www.guidegen.org

> "A patient can cancel his/her appointment. The doctor's office and the patient will get a cancellation e-mail. The patient must be logged-in first in order to make the cancellation."

As soon as the changes are applied to the requirement, GuideGen analyzes the changes and determines the relevant parts, i.e., those that impact an acceptance test linked to the changed requirement. In our example, the addition of the subject "patient" and of a complete sentence are relevant, while the added words "and" and "the" are irrelevant. Based on this analysis, GuideGen automatically generates guidance in natural language for the testers about how to change the associated acceptance test. The guidance consists of one suggestion per relevant changed element. In our example, the following suggestions are generated:

1. Make sure that now the patient will get a cancellation e-mail. Add the steps or modify the expected results which verify this activity.

2. Add new steps or modify existing steps to verify that the patient must be logged-in first in order to make the cancellation.

The details about how the analysis works and the suggestions are generated can be found in [HBCG18].

The GuideGen tool displays the generated guidance to the requirements engineer (Fig. 5.2) who can now review the generated suggestions, mark those that she considers to be irrelevant or wrong and then let GuideGen notify all people who have subscribed to such notifications.

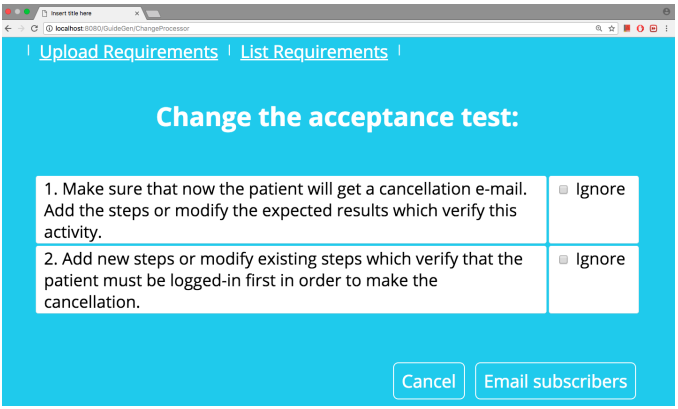


Figure 5.2: A list of suggestions generated by GuideGen

Notification works by an automatically generated e-mail, which contains the information about the affected acceptance test, the relevant suggestions on how to adapt that test and the summarized changes in natural language.

In the list of requirements, GuideGen also flags the acceptance test associated to the changed requirement with a warning sign, indicating that this test is outdated. When a test engineer receives a notification message from GuideGen, he can open the affected acceptance test case and edit it. GuideGen displays the editing window with the text of the test and the suggestions about what to change side by side (Fig. 5.3). When the tester saves his changes, GuideGen removes the warning flag from the list of requirements. More details about the GuideGen tool can be found in [HG18a].

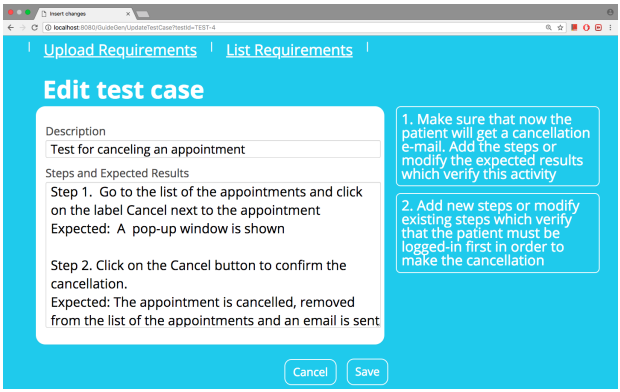


Figure 5.3: Suggestions that are shown to a test engineer in the affected test case

5.3 Research methodology

5.3.1 Research Goal and research questions

In [HBCG18] (see Chapter 3), we have evaluated GuideGen in terms of the correctness and appropriateness of the guidance generated by GuideGen. With a correctness score of more than 80% for real-world agile requirements and around 67% for traditional requirements, we found that GuideGen provides relevant and useful guidance for keeping acceptance tests aligned with their requirements. However, we did not assess the actual usefulness of GuideGen in practice. In this paper, our goal is to close this gap by performing a qualitative evaluation of GuideGen in terms of its perceived usefulness for practitioners and its

applicability to industrial software projects. From this goal, we derive four research questions:

RQ1: What are the current challenges with evolving requirements faced in industry?

RQ2: What are the benefits of GuideGen that can have a positive impact on the current challenges?

RQ3: How applicable is GuideGen to the current setup of real industrial projects?

RQ4: How useful is GuideGen for solving the currently faced issues with evolving requirements?

With RQ1, we assess the current situation with documenting and communicating changes in requirements and their tests and thus establish a *baseline*, against which the usefulness of GuideGen can be assessed. RQ2 investigates the benefits of GuideGen in order to assess its potential usefulness as perceived by industrial practitioners. RQ3 studies how well GuideGen can be used in the framework of the processes that practitioners currently use in their projects. Finally, RQ4 sheds light on the concrete usefulness of GuideGen for solving issues that practitioners are faced with in their current projects.

5.3.2 Study Design

To answer our four research questions, we performed a *qualitative experiment*[2], which consists of three parts.

[2] According to Kitchenham [Kit96], a qualitative experiment is "a feature-based evaluation done by a group of potential users who are expected to try out the methods/tools on typical tasks before making their evaluation".

1. *Introductory presentation.* The first author explained the GuideGen approach in a 20 minute talk to a group of business and technical practitioners. Then, a five minute video[3], which demonstrates the GuideGen tool, was shown to the audience, followed by a question and answer session of up to 30 minutes about both the GuideGen approach in general and the GuideGen tool. The first author then asked for requirements and test engineers who were interested to try out the tool and to evaluate GuideGen. We did this to ensure that only people who are interested and knowledgeable in the field of requirements engineering or testing participated in the experiment.

2. *Tool trial.* Although the participants already got familiar with the GuideGen tool by attending the presentation, we encouraged them to try it themselves, by using examples from their own projects. The tool trial lasted between 30 and 45 minutes.

3. *Interview.* After a five minute break, we conducted semi-structured interviews [Pat90] with the participants. Each interview consisted of two main parts: *an exploratory part* referring to the current issues and challenges faced when requirements change (RQ1) and *an evaluation part* about the usability and applicability of GuideGen (RQ2-RQ4). The interviews lasted between sixty and ninety minutes.

When there were more than two participants available or the presentation was held after the working hours, the first author came back the day after, otherwise the tool trial and interviews were held right after the presentation. The first author visited seven of the ten companies

[3] The video is available at https://tinyurl.com/y9leos5e

in person. For three companies (C6, C7 and C9 in Table 3.3) the presentation was held via Skype and only to the participants who were actually interviewed.

Demographics

The study is based on 13 interviews with 23 participants from ten companies. The companies are distributed over five countries in Europe: three companies are located in Switzerland, four in Serbia, one in Germany, one in Denmark and one in the Netherlands. We selected the companies by convenience sampling, using our personal contacts. We only selected companies that have documented requirements and acceptance tests, written in natural language. An overview of the companies is shown in Table 5.1.

All participants have between five and seventeen years of experience in IT and at least three years of experience in the requirements engineering and/or testing field. An overview of the participants and their teams is provided in Table 5.2. In total, we interviewed twelve test and quality assurance engineers, five requirements engineers and six industrial practitioners who are doing both requirements engineering and acceptance testing. We also present the size of the teams and the software process model applied within the teams, since that may impact the way that changes are communicated and documented, indirectly influencing the applicability of the approach.

Table 5.1: An overview of the companies

Company	Domain of activities	Country	# of employees
C1	Power industry	Serbia (International)	1000 (10000 worldwide)
C2	Access-control solutions	Switzerland (International)	1000 (10000 worldwide)
C3	Distribution Solutions	Switzerland (International)	500 (2500 worldwide)
C4	Enterprise Content Management	Switzerland (International)	50 (7000 worldwide)
C5	IT project and product management	Serbia (Germany, India)	20 (500 in total)
C6	Online marketing and online shopping	Germany (International)	1200 (3000 worldwide)
C7	Content Management Systems	Netherlands (International)	60 (600 worldwide)
C8	Software testing services	Serbia (Austria, Switzerland)	120 (500 in total)
C9	Data and Analytics	Denmark (Sweden)	250 (1000 in total)
C10	Website and mobile development	Serbia (Germany)	110 (350 in total)

Table 5.2: An overview of the participants and their teams.

Participant/ Company	Inter-view ID	Role	Years in IT / Years in RE or QA field	Team structure	Team size	Process model
P1/C1	I1	Test manager (QA)	8 / 8	cross-functional, co-located	10	Agile (Scrum)
P2/C1	I1	Test manager (QA)	5 / 5	cross-functional, co-located	15	Agile (Scrum)
P3/C1	I2	Subject matter expert (RE)	10 / 3	cross-functional, co-located	15	Traditional, Agile (Scrum)
P4/C1	I2	Test lead (QA)	5 / 5	cross-functional, distributed*	5	Traditional, Agile (Scrum)
P5/C1	I2	Subject matter expert (RE)	10 / 4	cross-functional, co-located	15	Traditional, Agile (Scrum)
P6/C1	I3	Senior test engineer (QA)	10 / 5	cross-functional, distributed*	30	Traditional, Agile (Scrum)
P7/C1	I3	Senior test engineer (QA)	12 / 5	cross-functional, distributed*	30	Traditional, Agile (Scrum)
P8/C1	I4	QA team lead (QA)	7 / 7	cross-functional, distributed*	20	Traditional
P9/C1	I4	Senior test engineer (QA)	5 / 5	cross-functional, distributed*	20	Traditional
P10/C2	I5	Product owner (RE)	10 / 10	mixed, co-located	8	Agile (Scrum)
P11/C2	I5	Senior test analyst (QA)	12 / 12	mixed, co-located	8	Agile (Scrum)
P12/C3	I6	QA manager (QA)	12 / 11	mixed, distributed	10	Traditional
P13/C4	I7	Head of operation (RE, QA)	17 / 17	mixed, distributed	1-5	Traditional
P14/C4	I7	Consultant (RE, QA)	17 / 17	mixed, distributed	10	Traditional
P15/C5	I8	Product manager (RE, QA)	6 / 4	mixed, distributed	4-5	Agile (Scrum)
P16/C6	I9	Senior Agile coach (RE, QA)	12 / 10	mixed, co-located	10-20	Agile(Scrum)
P17/C7	I10	Solution consultant (RE)	10 / 4	mixed, distributed	10	Agile (Scrum, Kanban)
P18/C8	I11	Agile coach (RE, QA)	12 / 10	mixed, co-located	3-9	Agile (Scrum)
P19/C8	I11	Test manager (QA)	5 / 5	cross-functional, distributed	10	Agile (Scrum)
P20/C8	I11	Test manager (QA)	6 / 6	cross-functional, distributed	14	Agile (Scrum)
P21/C9	I12	Solution architect (RE)	6 / 4	mixed, distributed	15	Agile (Scrum)
P22/C10	I13	Software developer (QA)	6 / 2	mixed, distributed	8-10	Agile (Scrum)
P23/C10	I13	Team lead (RE, QA)	6 / 2	mixed, co-located	10	Agile (Scrum)

*located in different buildings within the same city.
RE - requirements engineers, QA - test and quality assurance engineers.

Data collection and analysis

All experiments were conducted in December 2017 by the first author. The majority of experiments were face-to-face, except for companies C6, C7 and C9 that were conducted via Skype, due to their location. When the presentation and the trial of the tool were not done in person but via Skype, we shared the screen and the participants dictated or copied in Skype chat the text that should be inserted into the tool. In such a way the participants P16, P17 and P21 could see how the tool performs using their own data, just as other participants who tried the tool.

The interview instrument[4] was prepared in advance and has been discussed with several researchers specialized in qualitative studies and/or requirements engineering. In addition, we conducted a pilot interview with one personal contact from industry before starting the interviews reported in this study. Six interviews were done in pairs, two were conducted with three people at the same time and the remaining five were done with one person, as shown in Table 5.2. All interviews were conducted in English.

The interview instrument contains six parts, with questions referring to: (1) the interviewee's profile, (2) the profile of the company, (3) the software process model applied within a team, (4) how requirements and acceptance tests are documented and linked, (5) how requirements and acceptance tests are updated, and (6) the evaluation of GuideGen in terms of its usability and applicability. When asked about

[4] https://tinyurl.com/y9hfgwn7

how requirements and acceptance tests are managed, documented and updated (parts 3, 4 and 5), the interviewees explained also the issues they are faced with, thus answering RQ1. In part 6, we asked how GuideGen could influence the communication (6.5) and the way of working with requirements and acceptance tests (6.4), thus learning about the perceived benefits of GuideGen (RQ2). The applicability of GuideGen (RQ3) was assessed by asking whether the current implementation of GuideGen fits to the interviewees' current projects (6.6-6.9). Finally, we learned about the usefulness of GuideGen by directly asking the interviewees for their opinion (6.1, 6.2 and 6.8).

For analyzing the data, all interviews were recorded and later transcribed. The answers from all participants for each of the questions were grouped in order to be able to compare them. In such a way it was possible to quantify the answers to certain topics during the analysis.

5.4 Results

In this section, we present the results of our study, grouped by our four research questions.

5.4.1 Current issues with changing requirements (RQ1)

We first investigated the current challenges with changing requirements faced by the industrial practitioners. Although previous research [HBCG16], [BRB+14], [LB14] already has identified the most

common challenges, we needed to assess the current state of practice as seen by the participants of our study, in order to set up a baseline for the subsequent evaluation of the usefulness and applicability of GuideGen. The results show that the challenges identified in previous work are also perceived by the participants of our study. The most frequently mentioned challenges concern communication, documentation and tools.

Communication issues. When asking about the difficulties faced when requirements evolve, communication issues were explicitly mentioned by ten participants (P1, P4, P6, P8, P10, P12, P14, P16, P21 and P22) and agreed by seven other participants who were interviewed together with one of those ten (P2, P3 and P5, P7, P9, P13 and P23). Not surprisingly, all seventeen participants work in distributed or cross-functional teams, except P16 who explained that the communication problems occur when: *"people are coming and leaving the company, they don't know all the communication channels that we use in the company or they don't know the whole domain"*.

The most frequently reported communication issues are: late- or non-communicated changes to test engineers (CI1)[5], passing incomplete information from requirements to test engineers (CI2) and inability to track changes due to different communication channels (CI3).

Several researchers and practitioners advocate the involvement of test engineers in all requirements related activities as a best practice [UKKD08], [HBCG16], [Adz09]. However, in many companies this practice is still not applied. P1, P4, P6, P12 and P14 stated that

[5]We assign identifiers to all issues so that we can refer to them later.

changes in requirements are not communicated on-time to test engineers due to excluding testers from requirements activities. P2 framed the problem as follows: *"They* [the requirements engineers and the top management] *don't understand how important it is and we don't have that mindset that product owners and test engineers are "brothers". They need to work together and write documents also together, to be sure that they are on the same page".*

Moreover, P21 and P22 said that sometimes changes in requirements are not communicated to testers at all: under time pressure, requirements engineers often communicate changes only to developers.

Another problem, mentioned by P6, P8 and P12, is passing incomplete information from requirements engineers to test engineers. This causes test engineers to apply incorrect changes to test scenarios. Test managers P8 and P9 mentioned that the quality of communicated information sometimes depends on the mood of requirements engineers. P8 explained: *"Sometimes it goes smoothly and we clarify everything, but sometimes if they are in a bad mood we either don't hear from them at all, or we have unpleasant conversations, we don't get enough data and we cannot later do our job correctly".*

Further, P4 and P16 reported difficulties for test and requirements engineers to track changes when different communication channels are used at the same time, such as verbal communication, e-mails and chats.

Documentation issues are also present. The most frequently mentioned ones are: outdated documents (DI1), incorrect acceptance tests (DI2) and bad quality of requirements documents (DI3).

While there is research addressing the problem of outdated documents (e.g., [CKG12], [DWP15]), this is still a major issue in practice. P15 stated: *"The documentation is the first thing to suffer if there is no time or* [when there is] *a lot of stress or a lot of people involved"*. P7 explained that this is due to lack of time and a pressure from clients. The problem of outdated documents is also mentioned by P2, P8 and P12.

Incorrect acceptance tests are another issue that occurs mostly when testers are new and do not understand the context or when they are excluded from designing or modifying requirements, explained P5, P6, P11, P16 and P17. Another reason for having incorrect tests is that test engineers mostly rely on their own experience when writing or modifying tests. This is due to bad quality of requirements documents and poor communication between test and requirements engineers.

The most common issues with regard to the bad quality of requirements documents are: incompleteness or lack of context (P2, P6, P9, P10, P16, P17, P22), outdated requirements (P1, P3, P7, P8 P15, P12) and lack of traceability links between requirements and tests (P8, P14). Besides the lack of time, another reason for outdated, incomplete and non-traceable requirements documents are lack of knowledge with regard to change impact and risk (P1, P16, P17) and preference for verbal communication (P10, P11).

Issues with the currently used tools were also reported frequently: usage of too many different tools for document management (TI1), complexity of the tools (TI2) and inconvenient notification system (TI3).

Table 5.3 shows the reported tools used for requirements and test document management. The most common tools for managing low-level or technical requirements are TFS and Jira. P12 and P22 said that they use Polarion for this purpose. For high-level requirements practitioners use Microsoft Word, Microsoft Excel or Confluence. The tools for managing manual acceptance tests are TFS, TestLink, Jira, QA complete, iScrum, Google Docs, Tosca, Polarion, Excel and Test rail.

Using too many different tools and having requirements and tests in various places is reported to be challenging (P2, P5, P16). For instance, P5 explained that they first get high-level requirements from their clients in an Excel document and then, based on that document and discussions with the clients, they create functional specification in Word, as an internal, high-level requirements document. From the functional specification they derive user stories for development, which is their technical requirement specification. P5 stated: *"Sometimes you really need to be experienced to quickly find information that is needed. You need to know what is the level of detail in order to know where to look for the information"*.

Another problem is the complexity of the current tools. For instance, in TFS or TestLink too much "meta-work" is needed to get basic information, such as a traceability link or a history of changes, as reported by P2 and P15. P15 stated: *"Too many clicks are needed and meta-steps in order to create a test case or to see the link between a test and its requirement. This is why I use Google Docs for smaller projects, but I am aware that this is not sustainable"*.

Table 5.3: Tools currently used for managing requirements and acceptance tests documents

Tool	Document type	Number of participants	Participants' ID
TFS	Low-level requirement	11	P1-P9, P13, P14
	Acceptance test	10	P1-P9, P13
	Acceptance criteria	2	P1, P2
Jira	Low-level requirement	10	P10, P11, P15-P21, P23
	Acceptance criteria	3	P18, P21, P23
Polarion	Low-level requirement	2	P12, P22
	Acceptance test	2	P12, P22
Microsoft Word	High-level requirement (functional spec.)	10	P3-P9 P13, P14, P17
Excel	High-level requirement	7	P3-P9
	Acceptance test	1	P14
Confluence	High-level requirement	2	P13, P14
TestLink	Acceptance test	9	P3-P9, P16, P15
QAComplete	Acceptance test	2	P10, P11
iScrum	Acceptance test	2	P1, P2
Google Docs	Acceptance test	2	P15, P16
Tosca	Acceptance test	2	P19, P20
Test rail	Acceptance test	1	P17

The most common problem reported with regard to the notification system in TFS is that a user must subscribe for every feature individually and personally, and there is no possibility that a user subscribes other users to a feature of interest. When there are too many features, practitioners sometimes miss to subscribe and are not notified about important changes.

Another issue with the notification system in the currently used tools is spamming, as reported by P6, P15, P22 and P23. In case of numerous changes practitioners get too many notifications and, therefore, some changes are missed or simply ignored.

5.4.2 Benefits and Strengths of GuideGen (RQ2)

After discussing the current challenges, we assessed which of them GuideGen can mitigate. We asked the participants how using GuideGen would influence their work. From their answers, we identified seven benefits of GuideGen (labeled B1-B6) which we present below.

B1. Faster communication of changes. Participants who work in cross-functional or distributed teams saw a benefit in the mechanism for communicating changes provided by GuideGen. For instance, P20 reported: *"It can also shorten time for the communication of changes and just to keep test documents aligned"*. Moreover, this benefit is recognized also by the participants who work in co-located environment, such as P1 and P10. For instance, P10 explained: *"In the companies*

where people don't sit next to another the tool would help to communicate changes more quickly". This benefit directly addresses the issue with late communication of changes (CI1) and indirectly the issue with writing incorrect acceptance tests (DI2).

B2. Changes no longer missed or overseen by testers. Several participants stated that with GuideGen it would happen much less that changes are communicated badly or not on-time. P7 explained: *"A lot of things would not pass by us. I would get the information about every change. That would make the process of maintaining the change and documents easier"*. P11 also saw the following benefit: *"If I am away for a business trip or on holidays, we would not miss (...) changes* [while being absent]*"*. This again refers to CI1 and to passing incomplete information from requirements to test engineers (CI2).

B3. Acceptance tests updated faster and easier. Participants stated that suggestions, automatically generated e-mails and warnings would result in faster and more timely updates of acceptance tests. For instance, P21 explained: *"It would be definitely faster and smoother to communicate changes and to apply the changes, because we would not need to think that much what to do, if the system would propose the changes based on the updates in the requirement. It helps that the whole process goes faster"*. The participants P1, P3, P4, P5, P8, P13, P15 and P20 also made similar statements.

Updating acceptance tests would not only become faster with GuideGen, but also easier. As P2 explained: *"It will be much easier, because we would have alerts for the changes and we would really use it to know how to change test cases and not only that the change is there"*. Participants P6, P7, P20 and P23 made similar statements.

By improving the process of communicating and understanding changes (B3), GuideGen addresses the issues CI1, CI2 and CI3. If industrial practitioners apply the changes suggested by the tool, that would positively affect the issue with outdated documents (DI1).

B4. Requirements and acceptance tests kept closely together. In some companies, requirements and acceptance tests are stored and managed in different tools, thus keeping them and their changes easily traceable becomes challenging. For instance, P5 explained: *"If I change something in a requirement, currently testers don't have a clue that something is changed. I think it's good to keep requirements and acceptance tests together and that changes are visible to everybody"*. This benefit directly addresses the issue with using too many tools for requirements and test management (TI1) and indirectly the issue with the quality of requirements (DI3), since GuideGen establishes traceability links between requirements and tests. P15 and P19 made similar statements.

B5. Simple to use. The GuideGen tool misses many features that other tools have (see 5.4.3). P15 and P23 perceived this as a benefit and stated that due to its simplicity they would use GuideGen for smaller projects rather than TFS or Jira. P15 explained that for smaller projects she uses Google docs in order to avoid the complexity of TFS and Jira and that GuideGen provides much better possibilities for document management than Google docs. This partially addresses the issue with the complexity of the current tools (TI2).

B6. Serving as a reminder for updating. P18 rated the way of communicating changes with GuideGen as follows: *"It would be a good*

reminder and a good summary". P12 rated the warnings as follows: *"I see it as a good reminder for me not to forget to update a test case"*, while P11 emphasized: *"E-mails can easily be ignored, but warnings in the app itself will remind us that the documents are non-aligned and that there has been a change in the requirement. The warning cannot be ignored forever".* This partially solves the issues DI1 (outdated documents), because test cases will not stay outdated, and TI3 (inconvenient notification system).

5.4.3 Applicability of GuideGen to real industrial projects (RQ3)

In order to assess how applicable GuideGen is to the current setup in the companies, we specifically asked the participants if they would use the GuideGen tool for their projects if it were available. Two participants said that they would use it for smaller projects as is (P15, P23), while 14 (P5, P9-P14, P16-P22) said that they would use GuideGen if it were available as a plug-in for their existing tools. The remaining 7 participants (P1-P4, P6, P7, P8) said they would use it, but would need more features. Two of them (P6 and P7) mentioned that GuideGen could be a standalone application, but then the changes should be automatically replicated to the currently used tools, such as Jira or TFS.

In the following two sub-sections we present the major concerns towards the applicability of GuideGen and what features the participants missed in the GuideGen tool.

Major concerns about the applicability of GuideGen

The majority of participants was concerned about the data migration effort when switching to a new tool, in particular, when the amount of documents is high.

Next, there are different types of requirements and acceptance tests, defined with different levels of detail, while GuideGen currently supports only one type of requirements and tests. In real projects, requirements can be specified as high level requirements, followed by more detailed functional specifications, epics, user stories, etc. Acceptance tests can be, for instance, internal acceptance tests which usually correspond to user stories or business acceptance tests which usually consist of a couple of functional tests, covering one business unit. Furthermore, some companies do not have acceptance tests, but only acceptance criteria defined within user stories, explained P18, P20 and P23.

Further, participants were concerned about the bad quality of requirements in their company and how that may influence the performance of the GuideGen tool. Too short and incomplete sentences, too long sentences and grammatically incorrect sentences are often present in requirements. Changes in such sentences might cause that the tool generates incorrect suggestions. Another concern are outdated requirements that directly hinder the applicability of GuideGen.

Finally, the problem with spamming by sending too many e-mails due to frequent changes was another concern.

Features missing in GuideGen

We then specifically asked about the features that would be needed to make GuideGen applicable to the participants' current projects, but are missing in the current version of the GuideGen tool. The missing features mentioned in at least two interviews are presented in Table 5.4. They are mostly related to the workflow and status management or the usability.

Table 5.4: The missing features reported by at least two participants in two different interviews

Missing features	Participants
A possibility to see the outcome of the test execution (e.g. skipped, passed, failed, not started...)	P2, P4, P6, P12, P15, P19, P21
A possibility to see the status of a requirement (e.g. new, assigned, opened, finished...)	P6, P12, P13, P15, P17, P21
Tracking the history of changes in requirements and tests	P5, P7, P16, P20, P23
Personalize the user (having different accounts with different privileges)	P7, P9, P14, P16, P21
Different structure for keeping the documents (more folders for grouping different types of requirements, different projects or features)	P3, P5, P6, P12

Other features mentioned by at least one participant are: an improvement of the user interface for documenting test cases (P4), linking test cases to a specific acceptance criterion and not to the whole require-

ment or user story (P1), multiple languages support (P9), identifying which test case is actually impacted among many related ones (P15), identifying which concrete step in the test case is affected (P11), a possibility to make a test plan and not only scenarios (P2), including bugs in the system besides requirements and test cases (P2), ability to change and adapt the generated suggestion (P5), an automated analysis of changes in diagrams and images (P8), proposing manual checking instead of generating suggestions in case when a requirement is changed more than 60% (P8), a possibility to send a message to all subscribers on Slack instead of sending e-mails (P16), a central place to maintain boards like in Jira and estimation and prioritization tools (P17).

The current version of the GuideGen tool is limited to a one-to-one relationship between requirements and acceptance tests [HG18a]. To assess the severity of this limitation, we explicitly asked about what relationship is the most common in practice. Two participants stated that they mostly have 1:N, another two reported 1:1, one participant reported M:1 and the remaining participants said that they have an M:N relationship. Two of those who reported M:N explained that due to lack of time they often have 1:1, but they aim at having multiple tests per requirement in the future.

5.4.4 Quantitative evaluation of usefulness (RQ4)

We performed a short quantitative assessment of GuideGen by asking the participants to evaluate its usefulness on a five point scale, from 1 (totally useless) to 5 (extremely useful). We asked about the usefulness

of GuideGen both in general and for the participants' companies. The summary of the results is shown in Figure 5.4.

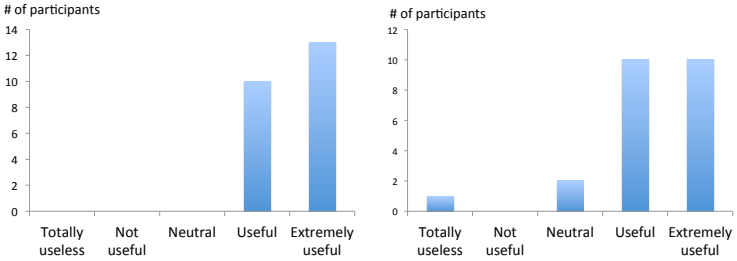


Figure 5.4: The usefulness of GuideGen in general (left) and for the company (right)

The participants P1-P5, P8-P11, P14, P16, P17 and P23 evaluated the approach as extremely useful and P6, P7, P12, P13, P15, P18-P22 as useful in general. When we asked for their work in their current company, the participants P1, P3-P5, P7-P9, P14, P15 and P23 rated GuideGen as extremely useful, P2, P6, P10-P13, P17, P19, P21, P22 as useful, P18 and P20 were neutral and P16 said that as a standalone tool it would be totally useless for his work. P18 explained that he is neutral due to not enough time to rate the usefulness of GuideGen thoroughly. P20 said that in his team acceptance tests are specified and updated by the clients and, therefore, GuideGen would not affect his work. P16 explained that GuideGen is useless for his work as a standalone tool due to challenges faced when switching to yet another tool. However, he said he would rate GuideGen as extremely useful if it were implemented as a plug-in for Jira.

We then asked our participants to rate the usefulness of the suggestion

feature (i.e., the suggestions generated by GuideGen about how to change an affected acceptance test). We gave them two scenarios, (1) the generated suggestions are correct in 98% of all cases and (2) they are correct in 80% of the cases, and asked them to rate the usefulness on a scale from 1 (totally useless) to 5 (extremely useful) for both scenarios. The results are shown in Figure 5.5.

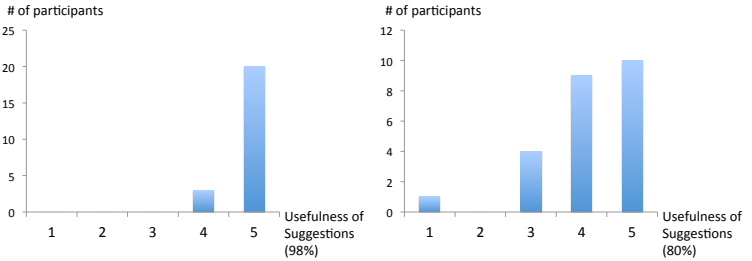


Figure 5.5: The usefulness of the suggestions when they are 98% correct (left) and 80% correct (right). 1 - totally useless, 2 - useless, 3 - neutral, 4 - useful, 5 - extremely useful

In scenario 1 where 98% of the generated suggestions are correct, all participants rated this feature as useful or extremely useful. In scenario 2 with 80% correct suggestions, only one participant (P19) said that suggestions would be useless at this level of correctness. Four participants (P5, P12, P18 and P21) gave a neutral rating and explained that it might be confusing to get 20% wrong suggestions. The other 19 participants rated the suggestion feature to be useful or extremely useful even at a correctness level of only 80%.

In addition, we asked whether suggestions are needed or the summary of the changes would be enough. 18 participants explicitly stated that

they need suggestions, while the others stated that it might be enough to have only the summary, but having the suggestion feature would nevertheless be useful and "nice to have".

5.4.5 Threats to Validity

Every qualitative research is exposed to several threats to validity. In this subsection we describe the possible threats.

Construct validity This aspect of validity reflects to what extent the operational measures that are studied really represent what is investigated [RH08]. We are aware that the participants would need to use GuideGen for a longer period of time in order to provide a more rational evaluation of its usefulness. However, due to the limited availability of the participants and the complexity of their systems and documentation, this was not possible. Therefore, we designed our study in such a way that the participants first got familiar with GuideGen through the presentation and the video. Moreover, after that introduction they had the opportunity to try out the GuideGen tool themselves, using examples from their own work.

The interviews were conducted by the researcher who created GuideGen, which might cause politeness bias in the interviewees' answers. We addressed this threat by telling the interviewees that they should be objective and express all their doubts and concerns about GuideGen.

Another threat to construct validity are misunderstandings between interviewer and interviewees. In order to mitigate this threat, we tested

our interview instrument in a pilot interview. In the interviews, we clarified questions whenever we had the impression that an interviewee had misunderstood them. The interviewees were also free to ask questions at any time.

Internal validity refers to credibility of drawn conclusions in a study [RST02]. In order to limit threats to internal validity, a good research design is always of crucial importance [IK11]. Therefore, we carefully designed our interview questions, we discussed them with a group of researchers, improved them and performed a pilot interview with a requirements engineer, who is our personal contact in industry. The discussion with researchers regarding the interview design helped us to limit also the reliability threat. However, reliability threats regarding researcher bias cannot be completely ruled out, because the interviews were conducted by only one researcher.

In order to avoid selection bias, we avoided personal contacts when we performed the interviews for the study. In addition, we interviewed only industrial practitioners who were interested to try GuideGen and answer our questions.

External validity is concerned with the generalizability of the findings of a study [RH08]. In order to limit the threat to external validity, we tried to keep diversity in terms of structure of the chosen companies, their size, domain of responsibilities, processes applied within the companies and the way that requirements and tests are documented. In addition, we chose participants based on their roles, so that we discuss with the target group, which are practitioners responsible for requirements and test documentation. Furthermore, we tried to achieve some geographical diversity.

5.5 Discussion

In this section we discuss the study results and the lessons learned.

5.5.1 Current issues with changing requirements and how GuideGen can mitigate them (RQ1, RQ2)

Table 5.5 summarizes the most frequently reported issues and the benefits of GuideGen that can mitigate those issues. We found that late communication, miscommunication and even non-communication of changes as well as outdated documentation are still major issues in practice, which confirms the results of a previous study [HBCG16].

Some issues are mitigated by GuideGen in its current form. For example, e-mails and warnings created by GuideGen support a better change communication and ensure that changes are not overseen. However, in projects with a high change rate, the use of GuideGen might result in so many e-mails and warnings that the testers would feel spammed. Addressing this issue requires adaptations in GuideGen: the GuideGen notification system would have to be made configurable so that, for example, testers could opt for receiving only one message per day from GuideGen with a summary of all changes (as suggested by P15). The tool should also permit a user to subscribe other interested parties to a feature, as this is one of the major issues with the notification system of currently used tools.

There are also issues, particularly those concerning organizational culture, which cannot be solely addressed with a tool such as GuideGen.

Table 5.5: The reported issues and benefits of GuideGen that mitigate them.

Issue type	Issue description (ID)	Benefit ID
Communication	late-/non-communicated changes (CI1)	B1, B2, B3
	Passing incomplete information (CI2)	B2, B3
	Many communication channels (CI3)	B3
Documentation	Outdated documents (DI1)	B3, B6
	Incorrect test cases (DI2)	B1
	Bad quality of requirements (DI3)	B4
Tools	Too many different tools (TI1)	B4
	Complexity (TI2)	B5
	Inconvenient notification system (TI3)	B6

While GuideGen facilitates the process of keeping acceptance tests up-to-date (by providing guidance and reminders for test engineers), GuideGen cannot force test engineers to apply the changes. Consequently, GuideGen cannot solve the issue of outdated acceptance tests if test engineers do not update them.

The fact that GuideGen only has a small amount of features can be seen as both a weakness and an advantage. The weakness is that GuideGen cannot support projects having, for example, different levels

of requirements and tests with many-to-many relationships between them. GuideGen also does not provide any sophisticated means for managing requirements and tests or for tracking issues. The advantage is that GuideGen is easy to use and suffices for smaller projects where tools like Jira or TFS are reported to be too complex and overwhelming for the users.

5.5.2 Applicability of GuideGen (RQ3)

In terms of applicability, we found that (1) the GuideGen tool can be used as is for smaller projects, but cannot replace the existing tools used for complex projects, and (2) it is most applicable to low level requirements and acceptance test documents.

GuideGen as a standalone tool

GuideGen cannot replace currently used tools for three main reasons: (1) missing features, (2) the complexity of data migration and configuration setup, and (3) the reluctance of industrial practitioners to learn new tools and adjust their working procedures accordingly. The participants stated that GuideGen needs to provide more features, for example, for managing workflow, history of changes, and access rights, as well as a possibility to better organize documents. Next, in complex systems data migration is a difficult and expensive task. The participants P16 and P17 explained that they do not want to start using another tool when they already have all data and configurations set up

in the currently used tools. Finally, participants argued that learning how to use a new tool and adjusting to it also takes some time.

However, all participants see a lot of benefit in the suggestions created by GuideGen and in the proposed way of communicating changes. Therefore, many of them suggested that the tool should be a plug-in for the tools they already use. So they could profit from the core features of GuideGen without having to migrate their data or to adjust their processes to a new tool set.

Types of documents to which GuideGen is applicable

There are different types of requirements and acceptance test documents, written with different levels of detail, as explained in Section 5.4.3. GuideGen is most applicable to low-level requirements and functional acceptance tests, since their levels of detail match and changes in one document directly cause changes in another one. GuideGen is not applicable to comprehensive requirements documents whose level of detail and scope go way beyond the specification of functional and non-functional requirements, but also contain architecture details or process descriptions. Changes of those elements may not impact changes in acceptance tests.

GuideGen has been built for supporting manual acceptance testing. With a minor adaptation of suggestions and an integration within the existing frameworks for automated testing, the approach could also be applied to scenarios written using Gherkin syntax [Adz16], which is widely used for specifying automated acceptance tests. We have not

explored generating suggestions for how to change the test code when using automated, executable tests.

Since GuideGen analyzes textual requirements by using NLP methods, it is applicable only to textual documents. GuideGen cannot analyze models, graphs or images.

5.5.3 Usefulness of GuideGen (RQ4)

Davis [Dav89] defines perceived usefulness[6] and perceived ease of use[7] as the fundamental determiners of user acceptance. In this study, we concentrated on perceived usefulness. Ease of use was addressed indirectly with question 6.8 in the interview instrument. Benefit B5 (see Sect. 5.4.2) indicates that GuideGen is easy to use. The results show that most participants in our study perceived GuideGen as useful or extremely useful. However, some participants were less positive when they evaluated the usefulness of GuideGen for their own current work. This is, for instance, due to non-frequent requirements updates in their companies, as well as their reluctance to migrate all their data into a new tool or to learn yet another tool.

The usefulness of GuideGen could be improved by implementing GuideGen as a plug-in for currently used tools in companies or by implementing missing features. With regard to the usefulness of the notification system, the e-mail generation mechanism in GuideGen should be adapted in order to avoid spamming.

[6] The degree to which a person believes that using a particular system would enhance his/her performance.

[7] The degree to which a person believes that using a particular system would be free of effort.

5.6 Related Work

Challenges in requirements and testing alignment are investigated by many researchers. For instance, our previous work [HBCG16] assesses challenges with writing and updating requirements and acceptance tests. Bjarnason et al. [BRB+14] identified the challenges in maintaining the alignment of requirements and verification and validation activities when requirements change. Their results are confirmed by Larsson and Borg [LB14]. The results from the exploratory part of this study confirm that the challenges identified in [HBCG16] and [BRB+14] are still present. In this study we, not only identify the current challenges with requirements and test alignment, but also investigate which of them GuideGen can mitigate.

In order to facilitate the requirements change management process and to establish automated traceability for change impact analysis, researchers propose information retrieval (IR) [LMOP12], [MMS05], [ESSD13] and natural language processing (NLP) [ASG+15a], [CCTF17] methods. However, all these methods investigate how changes in requirements impact code or other requirements; they do not consider acceptance tests. Moreover, they only identify which documents are affected, but not how they should be changed, despite the recognized need for suggestions about how to handle requirements changes [NdlVS13]. GuideGen provides concrete suggestions on how to adapt acceptance tests based on the analyzed changes in requirements.

Many researchers (e.g. [BUEB15], [RTCT07], [MM08], [HS12]) studied the usage of acceptance tests for specifying requirements. This is

particularly applied in agile projects, in which acceptance criteria are used for specifying details of requirements. However, replacing requirements by acceptance criteria is not widely accepted in practice. This study also confirms that in many companies today, requirements documents are written, either as user stories or in traditional textual form.

In order to align requirements engineering and testing activities, researchers focused on defining formal models from informal requirements and automatically generating tests from these formal models [BEF11], [EGM+11], [PSM+09], [GR06]. For instance, Escalona et al. [EGM+11] provide an overview of the research with regard to generating tests from functional requirements. They concluded that in order to have a successful test generation, requirements would need to be specified in a much more formal way than they usually are in practice. Similarly, Post et al. [PSM+09] define a method for linking functional requirements to tests by creating a formal specification of requirements. Since practitioners are not keen in writing "code-like" or fully structured requirements, GuideGen aims at providing an approach that does not require practitioners to adapt the level of formalism in their requirements.

5.7 Conclusion

In this paper, we report on a qualitative assessment of the usefulness and practical applicability of GuideGen, a method and tool for keeping

requirements and acceptance test documents aligned via automatically generated guidance. The study is based on the data obtained from thirteen interviews in ten companies with twenty-three industrial practitioners. Based on the practitioners' opinions, we assessed how useful GuideGen is for meeting challenges with evolving requirements and how applicable it is for the current setup in the companies.

The results show that GuideGen is useful for improving (1) communication in cross-functional and distributed teams, and (2) the process of updating acceptance tests when requirements change. Having suggestions on how to adapt the acceptance tests based on the changes in requirements is evaluated as superior to only being informed about the changes in a requirement. Communicating changes via automatically generated e-mails and warnings is seen as a positive aspect of the approach, although some practitioners expressed concerns about a potential flood of notifications when changes occur frequently. In addition, some practitioners prefer verbal communication, but they see these features as a good reminder to adapt the affected tests.

In summary, our study confirms that the core ideas of GuideGen are useful in industrial practice. Concerning the current state of the GuideGen tool, the study shows that the tool is applicable for smaller projects as it is. For bigger projects, however, GuideGen would either have to be incorporated as a plug-in into existing document and project management tools, such as Jira or TFS, or it would have to be extended with numerous features in order to make it a useful and practical stand-alone tool.

In our future work, we plan to investigate how GuideGen can be implemented as a plug-in for some of the tools currently used in practice.

Further, we plan to address the limitation that GuideGen only allows one acceptance test per requirement. Supporting one-to-many relationships between requirements and acceptance tests introduces a new challenge: we will have to develop a method for identifying which acceptance tests are impacted by a change in the related requirement.

Chapter 6

GuideGen Overview and a Step Towards an Automated Change Impact Analysis

Original publication:
GuideGen: An Approach for Keeping Requirements and Acceptance Tests Aligned via Automatically Generated Guidance
S. Hotomski and M. Glinz
Information and Software Technology (IST) 110:17-38, 2019 ([HG19])

Abstract

Context: *When software-based systems evolve, their requirements change. The changes in requirements affect the associated acceptance tests, which should be adapted accordingly. In practice, however, requirements and their acceptance tests are not always kept up-to-date nor*

aligned. Such inconsistencies may introduce software quality problems, unintended costs and project delays.

Objective: *In order to keep evolving requirements and their acceptance tests aligned, we are developing an approach called GuideGen. GuideGen automatically generates guidance in natural language about how to adapt the impacted acceptance tests when their requirements change.*

Method: *We have implemented GuideGen as a prototype tool and evaluated it in two studies: first, by assessing the correctness, completeness, understandability and relevance of the generated guidance using three data sets from industry and second, by assessing the applicability and usefulness of the approach and the tool with 23 practitioners from ten companies.*

When a requirement having more than one associated acceptance test is changed, GuideGen currently generates guidance for all of them together. As a first step towards overcoming this limitation, we assessed how well existing methods for change impact analysis can identify the tests actually impacted by the changes in a requirement.

Results: *In the first study, we found that GuideGen produced correct guidance in about 67 to 89 percent of all changes. Our approach performed better for agile requirements than for traditional ones. The results of the second study show that GuideGen is perceived to be useful, but that the practitioners would prefer a GuideGen plug-in for commercial tools instead of a standalone tool. Further, in our experiment we could correctly identify the affected acceptance tests for 63% to 91% of the changes in the requirements.*

Conclusion: *Our approach facilitates the alignment of acceptance tests with the actual requirements and can improve the communication between requirements engineers and testers.*

6.1 Introduction

Keeping requirements and their acceptance tests aligned and up-to-date is a challenging task due to the additional effort required and insufficient communication of requirement changes [BRB+14], [HBCG16]. Mismatches between requirements and acceptance tests eventually lead to mismatches between stakeholders' expectations and the actual software behavior. In many projects this is discovered only late and companies are faced with unintended costs, delivery delays and unsatisfied customers.

In order to have consistent and up-to-date documents, many researchers try to automatically identify which documents are related to each other and which of them are *impacted* by a change [BGW13], [LMOP12]. However, there is little research about how to actually *update* impacted documents, although it would be beneficial to have guidance about what actions to perform [NdlVS13].

To close this gap in the state of the art, we are developing *GuideGen*, an approach for keeping acceptance tests aligned with evolving requirements. GuideGen analyzes changes in textual requirements and automatically generates guidance on how to modify affected acceptance tests. Our approach aims at both reducing the effort for aligning acceptance tests with the actual requirements and improving the communication between requirements engineers and developers/testers. We have demonstrated the feasibility of our approach by implementing a prototype tool for GuideGen.

By analyzing changed sentences and words in a requirement, we derive guidance in form of a set of concrete suggestions about what should be changed in the acceptance test(s) associated with a changed requirement. Note that GuideGen assumes that traceability links between requirements and acceptance tests exist. The GuideGen tool also provides an easy way for communicating changes and the generated guidance to all interested parties. The changes are communicated via automatically generated e-mails and warnings shown directly in the application.

We evaluated GuideGen with two studies. In the first study, we generated suggestions for 262 changes of real-world requirements in three companies. The suggestions were then evaluated in terms of correctness, completeness, understandability and relevance. In the second study, we performed a qualitative evaluation of the usefulness and applicability of GuideGen with twenty-three practitioners from ten companies.

The results from our qualitative evaluation showed that one of the major shortcomings of GuideGen is the limitation to only one acceptance test per requirement. Therefore, we improved GuideGen to support multiple tests per requirement. This introduced a new challenge of identifying which of the related tests is affected by a change in the related requirement. As a first step towards solving this challenge, we performed an experiment to assess whether existing approaches for change impact analysis between textual artifacts can be used for identifying the affected acceptance tests.

The paper is organized as follows. In the next section we summarize the main contributions of this paper. We then present our approach

in Section 6.3 and describe our tool in Section 6.4. In Sections 6.5 and 6.6, we present a quantitative and a qualitative evaluation of GuideGen. In Section 6.7 we describe an assessment of the existing methods for change impact analysis and their applicability to GuideGen. The results of the evaluations and assessment are discussed in Section 6.8. Related work is presented in Section 6.9 and Section 6.10 concludes.

6.2 Main contributions of the paper

This article is an extension of our REFSQ 2018 conference paper [HBCG18]. It provides a comprehensive description of GuideGen: the approach, the tool and two evaluations. To achieve this, we also include some material from three other publications about GuideGen [HG18a], [HG18b], [HBCG17].

The main contributions of this paper over our previous work are as follows:

First we present the improvements of GuideGen that we made based on the feedback received from the two evaluations described in [HBCG18] and [HG18b] (see Chapters 3 and 4, respectively):

(1) Better recall by also considering changes of numerical values. Although there were not many numerical values in our data sets, we learned that their changes affect the associated acceptance tests and

should be processed accordingly. By taking changes of numerical values into consideration when generating guidance, we reduce the risk of missing suggestions that are relevant to test engineers.

(2) Extended notification system. GuideGen now notifies subscribed users not only about modifications in requirements, but also about additions of new requirements and adaptations applied to acceptance tests.

(3) More features. By providing more features, we aim at extending the applicability of GuideGen to larger and more complex projects that, for instance, involve one-to-many relationships between requirements and acceptance tests documents, contain more complex document organization or different roles involved in the project.

In Section 6.4.5 we provide a summary of the extensions and differences between GuideGen versions.

Second, we assessed how well existing methods for change impact analysis between textual artifacts perform when applied between requirements and acceptance tests. In an experiment, we applied NARCIA [ASG+15a] and ImpRec [BWRR17] on a small data set from industry. We found that, when the correct input is given, NARCIA can correctly identify affected acceptance tests in more than 90% and ImpRec in around 63% of the cases. The experiment is described in detail in Section 6.7.

Finally, we provide more details about the GuideGen approach and the GuideGen tool, for example, about the rules for generating guidance, the architecture of the tool, and its user interface.

Combining the previous and the new contributions we provide a comprehensive overview of the GuideGen approach and tool.

6.3 The GuideGen Approach

In this section, we first provide an overview of the GuideGen approach. Then we discuss the relevant implementation details.

6.3.1 Approach overview

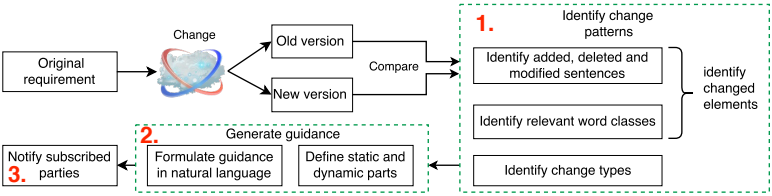


Figure 6.1: Overview of the GuideGen approach [HBCG18]

Figure 6.1 presents an overview of the GuideGen approach. As soon as a requirements engineer applies changes to a requirement and saves them, GuideGen performs the following three steps:

1. Identification of relevant change patterns: By comparing the old and the new version of the changed requirement, GuideGen identifies the elements that have been changed and their change types;

2. Generating guidance: In this step, GuideGen formulates suggestions in natural language on how to manage the changes;

3. Notification of subscribed parties: Finally, the changes can be communicated to the interested parties via e-mails and warnings.

We explain these three steps in the subsequent sub-sections, using the following example:

> A doctor can add new doctors to the group. ~~The addition of a new doctor must be first approved by the admin.~~ The admin and the doctor can modify personal data ~~and the status~~ of that ~~a~~ doctor. Only doctors can modify their status. The admin must be logged-in in order to modify personal data of a doctor.

Added words are green and underlined, removed words are red and struck through and black words are unchanged. The suggestions generated for these changes are presented in Table 6.1.

Table 6.1: The identified relevant change patterns with the corresponding guidance.

Relevant change patterns	Generated guidance
Deletion of the sentence "The addition of a new doctor must be first approved by the admin"	**Delete steps or their parts which verify that** *the addition of a new doctor must be first approved by the admin.*
Addition of the subject "doctor"	**Make sure that now** *the doctor can modify personal data of that doctor.* **Add the steps which verify this activity.**
Deletion of the object "status"	**Delete steps or their parts which verify that** *the admin can modify the status of a doctor.*
Addition of the sentence "Only doctors can modify their status".	**Add new steps or modify existing steps to verify that** *only doctors can modify their status.*
Addition of the sentence "The admin must be logged-in in order to modify personal data of a doctor".	**Add new steps or modify existing steps to verify that** *the admin must be logged-in in order to modify personal data of a doctor.*

Identification of Relevant Change Patterns

The first step in our approach is to identify relevant change patterns applied to a requirement.

A *change pattern* is characterized by the change type (add, delete, or modify) and the changed element (a whole sentence or a word). For instance, when the changed element is a whole sentence, the change pattern is "A sentence is added" or "A sentence is deleted". If the changed element is a word, an example of a change pattern is "A verb is deleted" or " A verb is modified".

Relevant change patterns are the ones whose changes require the acceptance tests to be adapted. In particular, relevant change patterns in our approach are the ones that directly or indirectly cause the change of some action, since acceptance tests contain a list of actions to be performed.

To identify the relevant change patterns, we first analyze whether a sentence is added, deleted or modified. Then we proceed by analyzing the relevant word classes (e.g. verbs, nouns, etc.) that have been changed in a modified sentence. Finally we classify each of the detected changes as relevant or irrelevant.

Identify added, deleted and modified sentences. The goal of this step is to identify whether a whole sentence has been added, deleted or its parts have been modified.

In order to do this, we first split the old and the new version of the updated requirement into sentences (oldReq and newReq in further text).

We then compare all the sentences from oldReq with the sentences from newReq by calculating the similarity between them. Based on the similarity, we determine whether the sentence is unchanged, added, deleted or modified.

Identify relevant word classes. After identifying sentences that have been added, deleted and modified, we proceed to analyze what changes were applied to the modified sentences. We identify word classes (e.g., noun, verb) in the sentence and then, for each of these classes, we determine its change type (add/delete/modify).

Identify change types. In this step we perform classifying identified changes into relevant and irrelevant. We consider a change to be relevant if it is likely to impact acceptance tests. Since acceptance tests contain a list of actions to be performed and as actions are generally expressed using verbs in English sentences, we consider verbs as the principal element of analysis in GuideGen. More concretely, we consider a change in a requirement to be relevant if it involves an addition, deletion or modification of a verb or of another relevant word class that relates to a verb.

Relevant word classes, besides verbs, are nouns, adjectives and numerical values. Changes of determiners, adverbs and prepositions are not taken into consideration, since we assume that they do not influence any actions and, therefore, do not have an impact on acceptance tests. If a whole sentence has been added, it is considered to be relevant only if it contains at least one verb.

Suggestions are generated only for the relevant change patterns, while other changes are considered to be irrelevant and are, therefore, not

further processed. As presented in Table 6.1, in our example, the following change patterns are considered to be relevant: (1) a deletion of the sentence "The addition of a new doctor must be first approved by the admin", (2) an addition of the noun "doctor", (3) a deletion of the noun "status", (4) an addition of the sentence "Only doctors can modify their status" and (5) an addition of the sentence "The admin must be logged-in in order to modify personal data of a doctor". Other change patterns, such as deleting the determiner "the" or conjunction "and" are irrelevant and not further processed.

In the following text we describe how we generate suggestions for the relevant change patterns.

Generating Guidance

The goal of this step is to generate suggestions about how to modify the affected acceptance tests so that they stay aligned with the changed requirements. An example of a suggestion is *Add new steps or modify existing steps to verify that only user can modify its status.*

In order to generate a suggestion we define its static and dynamic parts, based on the rules we formulated. All the rules for defining static and dynamic parts according to the change type are presented in Table 6.2.

The details about how we formulate the static and the dynamic parts of a suggestion are presented in Subsection 6.3.2.

Table 6.2: The rules for formulating suggestions. Static parts are in boldface, while dynamic parts are italicized.

Change patterns:	**The structure of suggestions:**
An addition/a deletion of a whole sentence	**Add/Delete steps or their parts which verify that** + *the whole sentence that has been added or deleted.*
An addition/a deletion of a non-auxiliary verb that is not in "ing" form and has at least one related subject	**Add/Delete steps or their parts which verify that** + *subjects with their determiners and adjectives + auxiliary verbs + adverbs + changed verb + prepositions + objects with their determiners and adjectives (sorted by word ID).*
An addition/a deletion of a non-auxiliary verb that is not in "ing" form and has no related subjects	**Add new/Delete steps or modify existing steps to verify that someone should** + *auxiliary verbs + adverbs + changed verb + prepositions + objects with their determiners and adjectives (sorted by word ID).*
An addition/a deletion of a non-auxiliary verb in "ing" form	**Add new/Delete steps or modify existing steps to verify that** + *adverbs + changed verb + words after the added verb.*
An addition/a deletion of a non-auxiliary verb in "ing" form	**Add new/Delete steps or modify existing steps to verify that** + *adverbs + changed verb + words after the added verb.*
Changing verb from positive to negative (e.g. does-> doesn't) or from negative to positive	**Make sure that now +** *subjects + changed verb + verbs related + prepositions + objects with their determiners and adjectives (sorted by word ID).*
A modification of a verb that has at least one related subject	**Modify steps which verify that** + *subject(s) + auxiliary verb + the old verb + prepositions + objects with their determiners and adjectives (sorted by word ID).* **Replace** + *the old verb* + **with** + *the new verb* + **in order to test that** + *subject(s) + auxiliary verbs + the new verb + prepositions + objects with their determiners and adjectives (sorted by word ID).*
A modification of a verb with no subjects related	**Modify steps which verify how to** *the old verb + prepositions + objects with their determiners and adjectives (sorted by word ID).* **Replace** + *the old verb* + **with** + *the new verb* + **in order to test that someone should +** *the new verb + prepositions + objects with their determiners and adjectives (sorted by word ID).*
An addition of a subject	**Make sure that now** *the added subject + the remaining words in the sentence excluding other subjects.* **Add the steps which verify this activity**.
A deletion of a subject	**Delete steps which verify that** *the deleted subject + the remaining words in the sentence excluding other subjects.*
A modification of a subject	**Modify steps which verify that** *the old subject + the remaining words in the old sentence excluding other subjects.* **Replace** + *the old subject* + **with** + *the new subject* + **in order to test that** + *the new subject + the remaining words in the new sentence until another non-related subject.*
An addition/a deletion of an object that has at least one related subject	**Add new/delete steps which verify that** *subject(s) with their determiners and adjectives + verb(s) + prepositions + added object with their determiners, adjectives and numerical values.*
An addition/a deletion of an object with no subjects related	**Add new/delete steps which verify that someone should** *verb(s) + prepositions + added object with their determiners, adjectives and numerical values.*
A modification of an object that has at least one related subject	**Modify steps which verify that** +*subject(s) with their determiners and adjectives + verb(s) + prepositions + the old object with its determiners and adjectives.* **Replace +** *the old object* **+ the new object + in order to test that +** *subject(s) with its determiners and adjectives + verb(s) + prepositions + the new object with its determiners, adjectives and numerical values.*
A modification of an object with no subjects related	**Modify steps which verify that someone should** + *verb(s) + prepositions + the old object with its determiners and adjectives.* **Replace +** *the old object* **+ the new object + in order to test that someone should +** *verb(s) + prepositions + the new object with its determiners, adjectives and numerical values.*
A modification of a noun: singular to plural	**Make sure that now there are more than one** + *the modified noun.*
A modification of a noun: plural to singular	**Make sure that now there is only one** + *the modified noun.*
An addition of an adjective	**Make sure that now** + *the related noun* + **should be** + *the added adjective + the noun.* **Modify the steps in the acceptance test by adding the adjective** + *the added adjective* + **in front of the noun** + *the noun.*
A deletion of an adjective	**Make sure that now** + *noun* + **is not** + *the deleted adjective + noun* + **any more**. **Modify the steps in the acceptance test by deleting the adjective** + *the deleted adjective.*
A modification of an adjective	**Replace** + *the old adjective + noun* + **with** + *the new adjective + noun* **in the acceptance test**.
An addition of a numerical value with no other numerical value related, is greater that 1 and the change is not from singular to plural	**Make sure that now there are exactly** + *numerical value + the related noun.*
A deletion of a numerical value with no other numerical value related, is greater that 1 and the change is not from singular to plural	**Make sure that now it does not have to be exactly** + *the numerical value + the related noun.*
A modification of a numerical value with no other numerical value related, is greater that 1 and the change is not from singular to plural	**Make sure that now there are** + *the new numerical value + the related noun* + **instead of** + *the old numerical value + the related noun.*
An addition of a numerical value with other numerical value related	**Make sure that now there can be also** + *the numerical value + the related noun.*
A deletion of a numerical value with other numerical value related	**Make sure that now there cannot be** + *the numerical value + the related noun* + **any more**.

Notification of Subscribed Parties

Our notification system allows requirements engineers to send an *automatically generated e-mail* to subscribed parties (in particular, testers) when a requirement has been changed. The message contains the previous and the updated version of the requirement, the generated guidance and the summarized changes.

Further, the notification system sends an automatically generated e-mail to requirements engineers as soon as test engineers apply changes to an affected acceptance test.

When a new requirement is added GuideGen also generates an-email that is sent to testers and other subscribers. The e-mail contains the ID, title and text of the newly added requirement and a note that a new acceptance test should be added accordingly.

Finally, users are notified about changes and mismatches between documents via *warnings* that are shown in the application whenever acceptance tests are not aligned to their requirements or when a new requirement is added.

The examples of e-mails and warnings are shown in the next section.

6.3.2 Implementation details

In this subsection we describe implementation details of the previously described steps.

Identifying added, deleted and modified sentences

For splitting a requirement into sentences we use an implementation of the Stanford sentence splitting algorithm [MSB+14]. The similarity between two sentences is calculated using an existing semantic similarity toolkit [RLB+13], called SEMILAR. In particular, we use greedy matching for word to word similarity that is based on WordNet. The reason for choosing this algorithm is that other algorithms, provided in SEMILAR, are too slow due to usage of LSA and LDA corpora.

Sentences are unchanged when the similarity between them is equal to one. If they are changed and a sentence in the oldReq does not have a corresponding one in the newReq so that the similarity score between them is greater than a given *modification threshold*, then this sentence is considered as deleted. When the similarity score between two sentences is above the modification threshold, these sentences are candidates for modified sentences. We choose the best match – a pair of sentences whose similarity score is the highest among other pair candidates. When we remove best matches, unchanged sentences and already identified deleted sentences from the oldReq and the newReq, there might be leftovers. The leftovers in the newReq are added sentences and the leftovers in the oldReq are deleted sentences. A flow diagram and a pseudo code of the algorithm are shown in Figure 6.2.

Figure 6.3 shows the calculated similarities between the old and the new version of the changed requirement.

The first sentence is eliminated from the further analysis because the similarity score is S(1,1) = 1. Since all scores calculated for the second

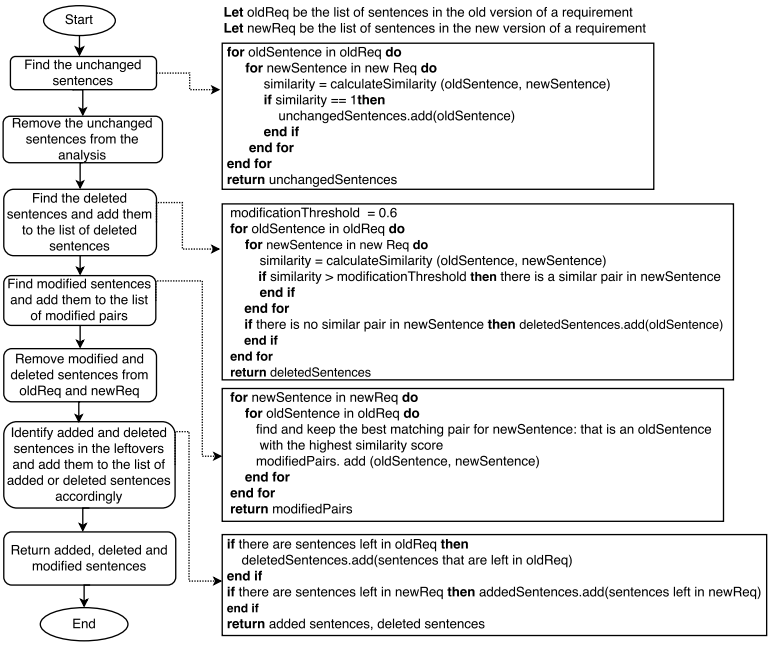


Figure 6.2: The algorithm for identifying added, deleted and modified sentences [HBCG18]

sentence, S(2,2) = 0.36, S(2,3) = 0.11 and S(2,4) = 0.5, are below the modification threshold (0.6), the second sentence in the oldReq is found to be deleted. The third sentence in the oldReq has two matching sentences in the newReq for which the similarity is above the modification threshold: S(3,2) = 0.86 and S(3,4) = 0.62. We choose the best match, in this case S(3,2). Therefore, the third sentence in the oldReq is modified to the second sentence in the newReq. The third and the fourth sentence in the newReq become leftovers. Since they are both in the newReq, we find these two sentences to be added.

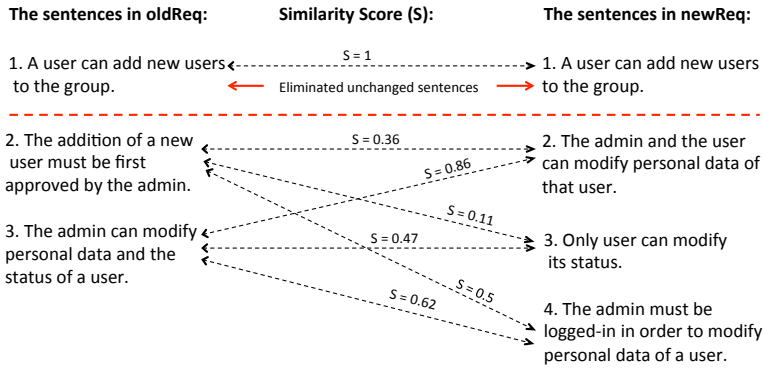


Figure 6.3: Calculated similarity scores for the sentences in the example [HBCG18]

Determining the modification threshold. We use a value of 0.6 for the modification threshold. We determined this value by experimentation: we took ten sentences from a requirements document and applied twenty change patterns to each of them; resulting in 200 comparisons. The value of 0.6 yielded the best results for identifying added, deleted and modified sentences in this sample. The evaluation of GuideGen confirmed that 0.6 is a good value for the modification threshold (see Sect. 6.5.2). Note that the data set used for determining the modification threshold was not used in the evaluation of GuideGen.

Managing enumerated sentences We treat enumerated sentences (i.e., sentences containing bullet points) by transforming them into plain sentences. For example, the tool transforms the sentence

"A user can insert: - name,

- surname".

into: “A user can insert name” and “A user can insert surname”. If a bullet point is added or deleted, the change is treated as an addition or a deletion of a plain sentence. For instance, if we add “- e-mail”, this change is treated as the addition of the sentence “A user can insert e-mail”. Otherwise, the addition of a noun that has no related verbs would be classified as an irrelevant change pattern and the addition of the noun “email” would not be further processed.

Identifying relevant word classes

For identifying word classes we use Google’s implementation of a globally normalized transition-based neural network model, called SyntaxNet [AAW+16]. Besides the word class, SyntaxNet determines the grammatical function (e.g., subject, object) for each word in a sentence, as well as dependencies between words which are represented as dependency numbers. We use these when defining static and dynamic parts of a suggestion, as it will be described in further text. Figure 6.4 shows an example of the output of SyntaxNet.

In order to identify whether words have been added, deleted or modified, we adapted the algorithm implemented in a text-based diff engine, called Text_Diff [CH]. Text_Diff detects changes at a phrase level. We process the output from Text_Diff so that we get the changes on a word level. We show this using the following modified sentence from our example:

“The admin and the doctor can modify personal data ~~and the status~~ of that ~~a~~ doctor”.

ID	TEXT	WORD CLASS	DEPENDENCY NUMBER	GRAMMATICAL FUNCTION
1	The	DET	2	det
2	admin	NOUN	4	nsubj
3	can	VERB	4	aux
4	modify	VERB	0	ROOT
5	personal	ADJ	6	amod
6	data	NOUN	4	dobj
7	and	CONJ	6	cc
8	the	DET	9	det
9	status	NOUN	6	conj
10	of	ADP	9	prep
11	a	DET	12	det
12	user	NOUN	10	pobj
13	.	PUNCT	0	punct

Figure 6.4: An example of the output of SyntaxNet [HBCG18]

When processing this sentence, the original Text_Diff algorithm detects the addition of the phrases “and the doctor” and “that” and the deletion of the phrases “and the status” and “a”, as follows:

> The admin <add> and the doctor </add> can modify personal data <del> and the status </del> of <add> that </add> <del> a </del> doctor.

We adapted the algorithm so that it detects additions and deletions of each word in these phrases:

> The admin <add> and </add> <add> the </add> <add >doctor </add> can modify personal data <del> and </del> <del> the </del> status </del> of <add> that </add> <del> a </del> doctor.

Generating Guidance

Deriving the rules for generating change suggestions We formulated the rules with informal experimentation and by considering typical sentence structures in requirements documents. We analyzed sentences from a requirements document written as free text and by analyzing publicly available user stories,(e.g. on Trello[1]).

In the free-text document we found and analyzed twenty requirements that have related acceptance tests. Since publicly available user stories often do not have acceptance tests related, we additionally used examples of user stories with acceptance criteria from the websites[2] that explain the best practices and are not necessarily focused on data only. None of these sentences nor documents are used when evaluating the approach.

Defining the static parts of a suggestion The static parts of a suggestion differ according to the previously identified change patterns. For instance, if a whole sentence has been added to a requirement, the static part of the suggestion is "Add steps or their parts which verify that", as presented in Table 6.2. Accordingly, if a whole sentence has been deleted, the static part of the suggestion is "Delete steps or their parts which verify that".

If a sentence has been modified, the static parts are formulated according to the modification type: whether a verb, subject, object, adjective

[1] https://trello.com/b/MGC4RpTZ/frictionless-data-user-stories

[2] e.g. https://medium.com/existek/acceptance-criteria-explanation-examples-and-template-82bdcde1d3c0

or a numerical value is added/deleted/modified or a noun is changed from singular to plural, etc. For instance, if a subject is added, the static parts of the suggestion are "Make sure that now +*{dynamic part}*" and "Add the steps which verify this activity".

Defining the dynamic parts of a suggestion The dynamic parts fill the gaps between the static parts. They differ according to the type of the changed element, as shown in Table 6.3.

Table 6.3: Words included in the dynamic part of a suggestion according to the changed element.

Changed element		Words included in the dynamic part
Sentence		changed element (all words in that sentence)
Noun	subject/ conjunction	changed element and all words that appear after that element, excluding other subjects in the sentence
	object/ conjunction	changed element, subjects with determiners and adjectives, verbs, prepositions with their objects
Verb		changed element, auxiliary verbs, subjects and objects with determiners and adjectives, prepositions with objects, adverbs
Adjective		changed element, related nouns
Numerical value		changed element, related nouns

Adding/Deleting the whole sentence: If a whole sentence has been added or deleted, the dynamic part contains all words in that sentence, as presented in Table 6.2. For instance, in our example, for the deleted sentence "The addition of a new doctor must be first approved by the admin", the dynamic part contains *all words* from that sentence, as presented in Table 6.1 in italic.

Adding/Deleting/Modifying a subject: When the changed element is a subject, the dynamic part contains that subject with its determiners

and adjectives, followed by the words that appear after the changed subject, as presented in Table 6.2. In our example, for the addition of the subject "doctor" the following words are taken for formulating the dynamic part: the determiner "the", which is directly related to the added noun "doctor", and the words that appear after the added noun in the new version of the changed requirement: "can modify personal data of that doctor". When sorted by word index, we formulate the dynamic part: *"the doctor can modify personal data of that doctor"*, as presented in Table 6.1.

If the modified sentence contains other subjects, besides the changed one, those are not included in the dynamic parts nor their related words. To identify the position of the words we use the word index (ID in Figure 6.4), which is provided by SyntaxNet.

Adding/Deleting/Modifying an object, a verb, an adjective or a numerical value: When the changed element is an object, a verb, an adjective or a numerical value, the dynamic part contains that element plus its related words. We identify the related words by analyzing word classes, grammatical functions and dependency IDs of words in the modified sentence, provided by SyntaxNet.

Related words for objects are:

- a verb whose index corresponds to the dependency ID of the object,
- a subject whose dependency ID refers to the index of the identified related verb,
- prepositions whose dependency IDs refer to the changed object and

- directly related numerical values.

We recursively include their related words in the dynamic part. If an object is related to another, main object by a conjunction, we identify the words that are related to the main object.

Related words for verbs are:

- directly related subjects,
- objects,
- prepositions and
- adverbs with their related words and corresponding indexes and dependency IDs.

Related words for numerical values and adjectives are the nouns that they directly relate to.

In our example, the deleted object “status” has a conjunction to the main object “data”, as shown in Figure 6.4. As previously explained, in this case we consider related words of the main object to be related words of the conjunction object. As the main object “data” has a related subject, we, therefore, apply the rule from Table 6.2 that refers to a deletion of an object that has at least one subject related. Therefore, we identify the verb “modify” and its auxiliary verb “can” as related to the deleted object “status”. The subject “admin” refers to the verb “modify” and has a related determiner “the”, so they are both classified as related words of the deleted object. The preposition “of” directly refers to “status” and it has the related noun “doctor” with its determiner “a”. The determiner “the” is directly related to “status”. The words are

ordered by the word index and the dynamic part is formulated as *"the admin can modify the status of a doctor"*, as presented in Table 6.1.

Managing several changes applied to a sentence When several relevant changes are applied to a sentence, we analyze which of the words are already processed and omit them from the further analysis. In such a way we avoid duplicated suggestions. For instance, when a verb and a noun are related and added to a sentence at once, we generate only one suggestion for the added verb and we omit the suggestion for the added noun, as the noun is already included in the first suggestion.

6.4 Tool Support

We have implemented our approach in a *prototype tool*. GuideGen is a web application written in Java and deployed on Apache Tomcat [WGC+04]. In this section, we first characterize the users for whom the GuideGen tool provides support. We then describe the typical actions that users perform when working with GuideGen. Next, we explain the architecture of the tool and its availability. Finally, we provide a summary of different versions of GuideGen.

6.4.1 GuideGen Users

On the one hand, GuideGen supports requirements engineers in maintaining the requirements of a system and in communicating

all changes of requirements to testers, developers and other interested parties on-time and with almost no effort. On the other hand, GuideGen supports testers, who maintain acceptance test documents, by providing them with guidance on how to modify impacted tests, so that they stay aligned and consistent with the modified requirements. In addition, by flagging all non-aligned acceptance tests, any stakeholder can easily see which acceptance tests are currently misaligned with their corresponding requirements—be it that tests do not exist yet or that they have not been updated after changes in the requirements.

6.4.2 Using GuideGen

Upon starting GuideGen, a user can log-in or sign up, depending on whether the user is already registered in the system or not. In the remainder of this sub-section, we describe twelve typical actions that requirements engineers, test engineers and other users may perform when working with the GuideGen tool.

Action 1. Logging to the system. Upon starting the application, the system show the index page with the log-in form and the link for signing up, as shown in Figure 6.5.

In case that a user is already registered in the system, she inserts credentials into the log-in form. If the inserted data are correct, the user is transferred to the next page. The content of the next page depends on the role of the user. We defined the following roles: *Requirements Engineer*, *Test Engineer* and *Other*. Based on the role of the user, the

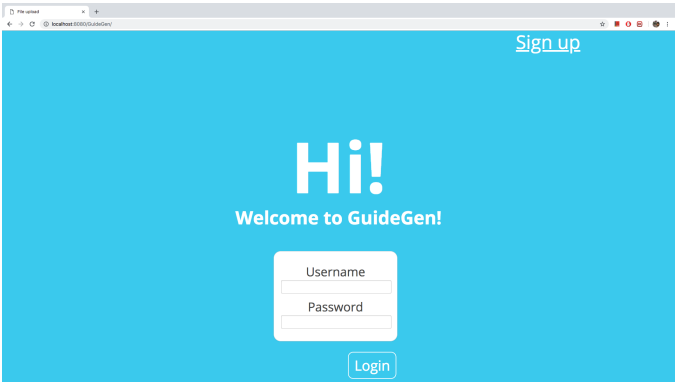


Figure 6.5: Index page: Log-in or Sign Up.

Table 6.4: The role-based privileges granted to a user.

The role of a user	Granted privileges
Requirements Engineer	can_edit_requirement_data, add_new_requirement, upload_list_of_requirements_and_tests
Test Engineer	can_edit_acceptance_test_data, add_new_acceptance_test, upload_list_of_requirements_and_tests
All (Requirements Engineer, Test Engineer, Other[3])	sign_Up, log-in, view_requirement_data, view_acceptance_test_data, filter_non_aligned_documents, choose_type_of_documents_to_list

system grants certain privileges for performing actions. The roles and privileges are shown in Table 6.4.

The table shows that only requirements engineers can edit or add new requirements and upload external, already defined requirements and tests. Further, only test engineers can edit or add new acceptance tests

[3]Other refers to developers, managers, architects and other stakeholders interested in the requirements and acceptance tests of a system.

and upload existing requirements and tests. The users with the role "Other" can only read the currently existing requirements and acceptance tests and their data though the system, without a possibility to edit or add data.

Figure 6.6 shows the content of the page to which requirements engineers are transferred after logging to the system, while Figure 6.7 shows the content for test engineers.

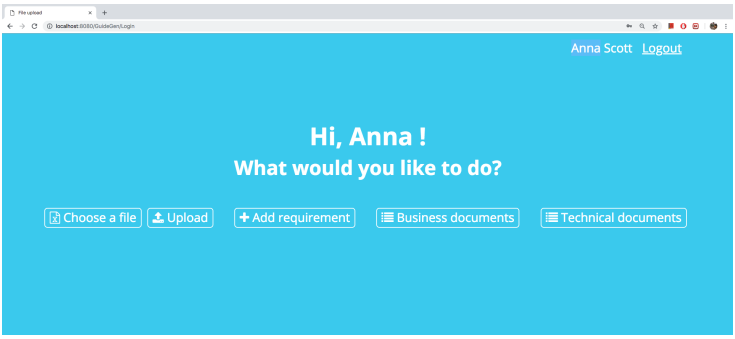


Figure 6.6: The form shown to requirements engineers after logging.

In Figure 6.7 is shown that test engineers cannot add a test case without any related requirement, as there is no button for adding an acceptance test. The usual practice is that new acceptance tests are added only after the requirements are added [HG18b], [HBCG16] and, therefore, we followed this concept in GuideGen.

Action 2. Signing up to the system. By clicking on the link "Sign up" (see Figure 6.5), a user is transferred to the page with the Sign up form, as shown in Figure 6.8.

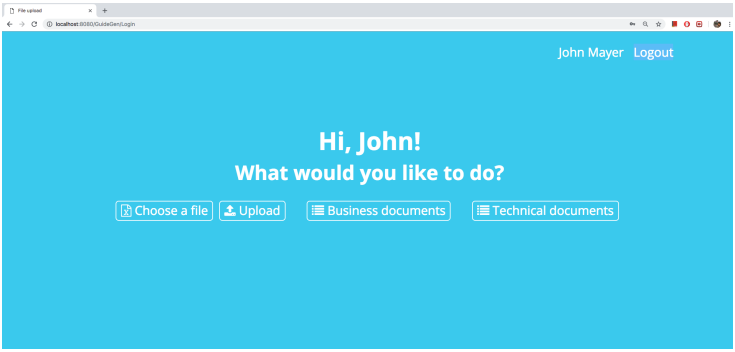


Figure 6.7: The form shown to test engineers after logging.

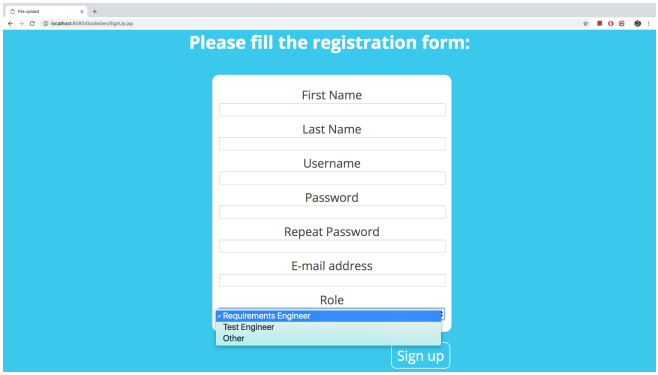


Figure 6.8: The Sign up form.

The user inserts her personal data and chooses one of the three, previously explained, roles.

Action 3. Uploading a new collection of requirements and their tests. When a user chooses to upload a new collection of requirements, she clicks on the button "Choose a file" (see Figure 6.6 and Figure 6.7).

In the current implementation of GuideGen, the user chooses an Excel file stored in the file system, from which the collection is loaded. This can be easily customized, so that GuideGen supports other file formats, besides Excel.

After uploading the requirements and their tests, the user can choose to view the content of one of the following folders: Technical, Business or All, as shown in Figure 6.9.

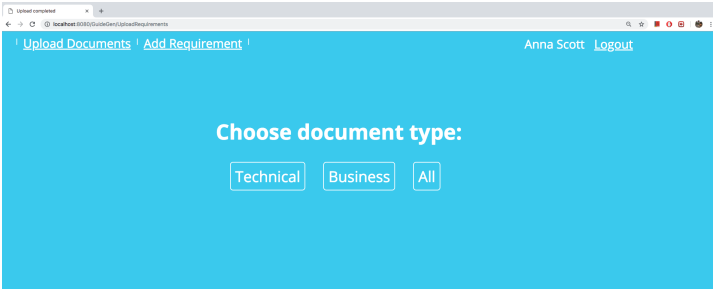
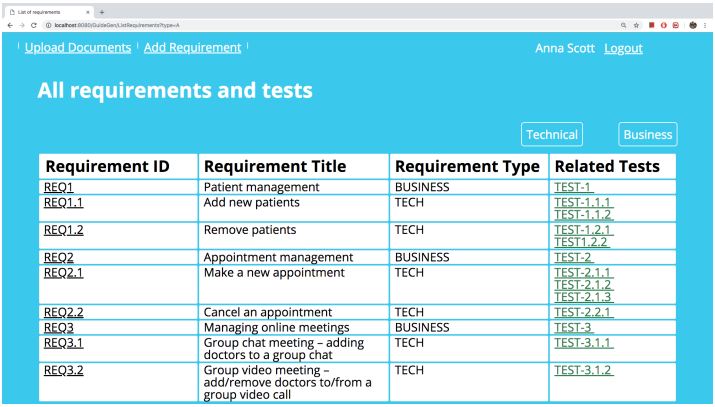


Figure 6.9: Folders with different types of documents.

Low-level requirements and acceptance tests are stored in the folder Technical, high-level requirements and tests are kept in the folder Business, while the folder All contains all requirements and tests that are currently present in the system.

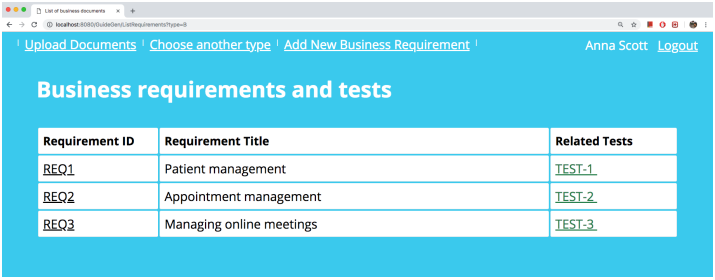
Figure 6.10 shows all the requirements and their tests that are currently present in the system, while business and technical requirements and their tests are presented respectively in Figure 6.11 and Figure 6.12.

Action 4. Adding a single requirement through the GuideGen form. In case that a new requirement is added by using the application,



Requirement ID	Requirement Title	Requirement Type	Related Tests
REQ1	Patient management	BUSINESS	TEST-1
REQ1.1	Add new patients	TECH	TEST-1.1.1 TEST-1.1.2
REQ1.2	Remove patients	TECH	TEST-1.2.1 TEST1.2.2
REQ2	Appointment management	BUSINESS	TEST-2
REQ2.1	Make a new appointment	TECH	TEST-2.1.1 TEST-2.1.2 TEST-2.1.3
REQ2.2	Cancel an appointment	TECH	TEST-2.2.1
REQ3	Managing online meetings	BUSINESS	TEST-3
REQ3.1	Group chat meeting – adding doctors to a group chat	TECH	TEST-3.1.1
REQ3.2	Group video meeting – add/remove doctors to/from a group video call	TECH	TEST-3.1.2

Figure 6.10: All documents present in the system.



Requirement ID	Requirement Title	Related Tests
REQ1	Patient management	TEST-1
REQ2	Appointment management	TEST-2
REQ3	Managing online meetings	TEST-3

Figure 6.11: List of high-level (business) documents.

GuideGen provides a user with a form that shows editable details about the new requirement that is being inserted. The option for adding a new requirement is shown to the requirements engineers only.

The form for adding a new requirement, is shown in Figure 6.13. In this case a new requirement is added from the initial page for requirements

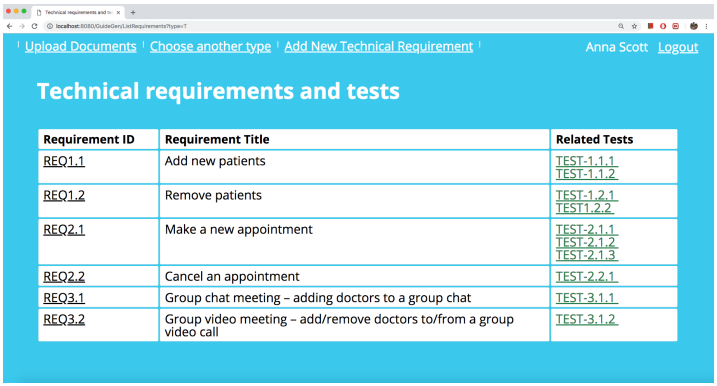


Figure 6.12: List of low-level (technical) documents.

engineers, shown previously in Figure 6.6 and, therefore, the requirements engineer chooses the type of the requirement. When a new business requirement is being added, the type "Business" will be pre-selected in the form for adding a new requirement. Accordingly, when a new technical requirement is being added, the type "Technical" is pre-selected.

All fields except “External link” are mandatory. As soon as the requirements engineer saves a newly added requirement, the tool marks the requirement with a warning sign, so that all stakeholders become aware of the change, as presented in Figure 6.14. The warning suggests that a new test should be added for the created requirement.

In addition, the tool notifies the subscribed test engineers by e-mail when a new requirement has been added. The content of the e-mail is shown in Figure 6.15.

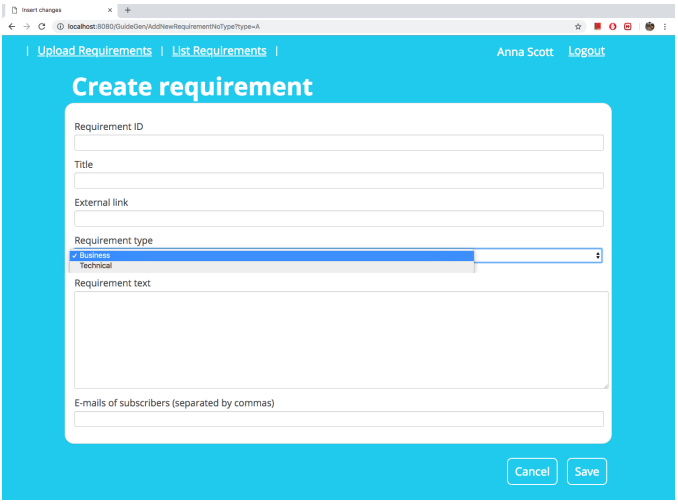


Figure 6.13: The form for adding a new requirement.

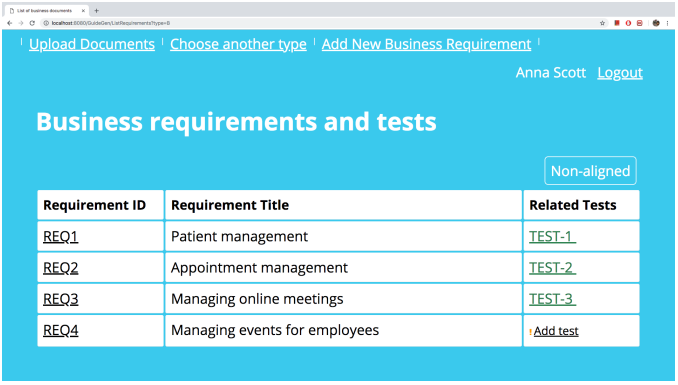


Figure 6.14: The warning after adding a new requirement.

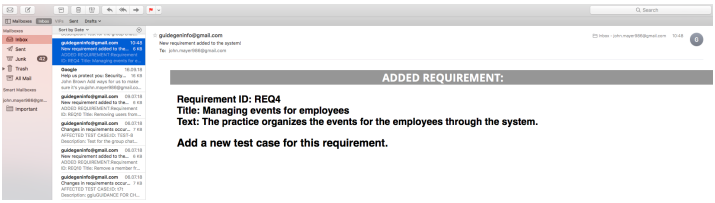


Figure 6.15: The e-mail generated when the requirement REQ4 is added.

Action 5. Adding a test case. By clicking on the warning link "Add test" (see Figure 6.14), a form for adding a test case is presented to a test engineer. The warning is visible to everybody, but the form opens only to test engineers. The form for adding a new test case is shown in Figure 6.16.

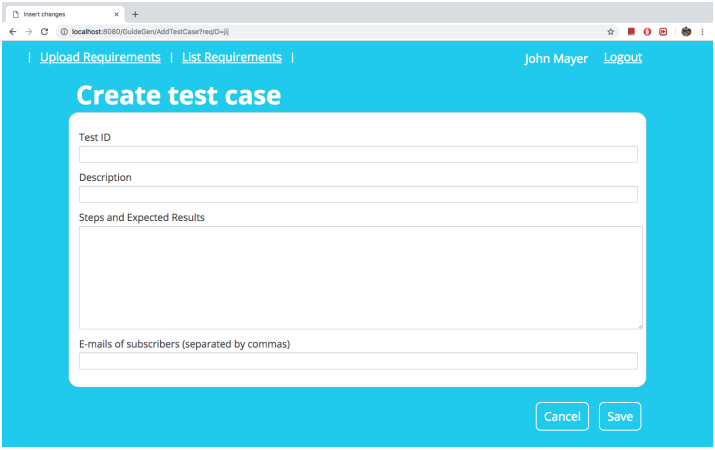


Figure 6.16: The form for adding a new test case.

By default the execution status of a newly added acceptance test is

"Not executed". When other data are inserted and saved, the warning about the missing test in the list of requirements and their acceptance tests is replaced by the ID of the newly added test.

Action 6. View the details of a requirement. When a requirements engineer clicks on the requirement ID in the list of requirements and their tests, a form with requirements details is shown, as presented in Figure 6.17. The form contains the information about the requirement document, such as ID, title, type, an external link and the text of the requirement. Next, the form contains the following meta-data: the creator of the requirement, the current status of the requirement (e.g. new, assigned, in development, etc.), responsible person for implementing the requirement and responsible person for testing the requirement.

All users are able to see the details of the requirement and its history of changes and to add subscribers, while only requirements engineers are able to edit the details and meta-data of the requirement. Therefore, the buttons for editing the requirement and its meta-data are shown only to requirements engineers.

Figure 6.18 shows the form with the details of the requirement from a perspective of a test engineer.

Action 7. Editing a requirement. Figure 6.19 shows a form for editing the information about the requirement. All data except the ID of a requirement can be modified.

When information other than the text of a requirement is changed, GuideGen saves the changes without performing further analysis.

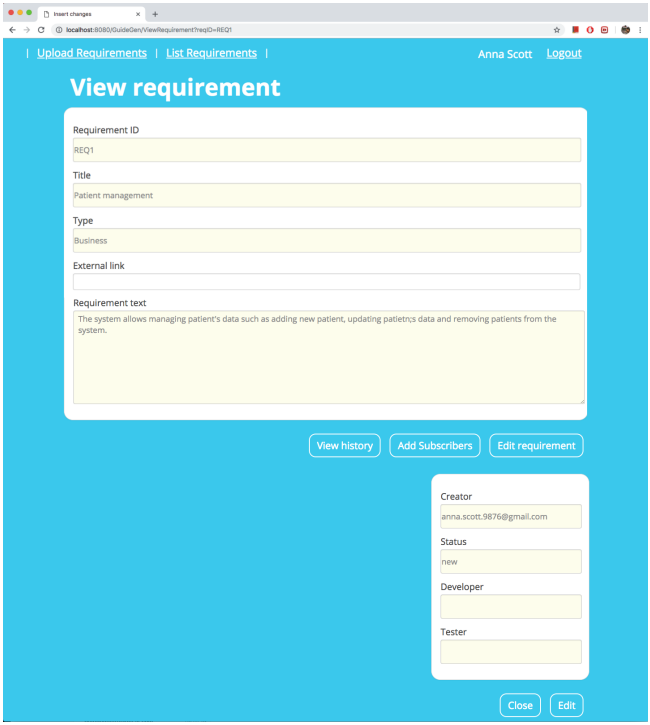


Figure 6.17: The form with details of a requirement from the perspective of a requirements engineer.

When the text of a requirement is changed, as soon as the changes are saved GuideGen performs the analysis of the changes, generates a list of suggestions and shows the suggestions to the requirements engineer. The list of the generated suggestions for changes applied to REQ3.1 is shown in Figure 6.20.

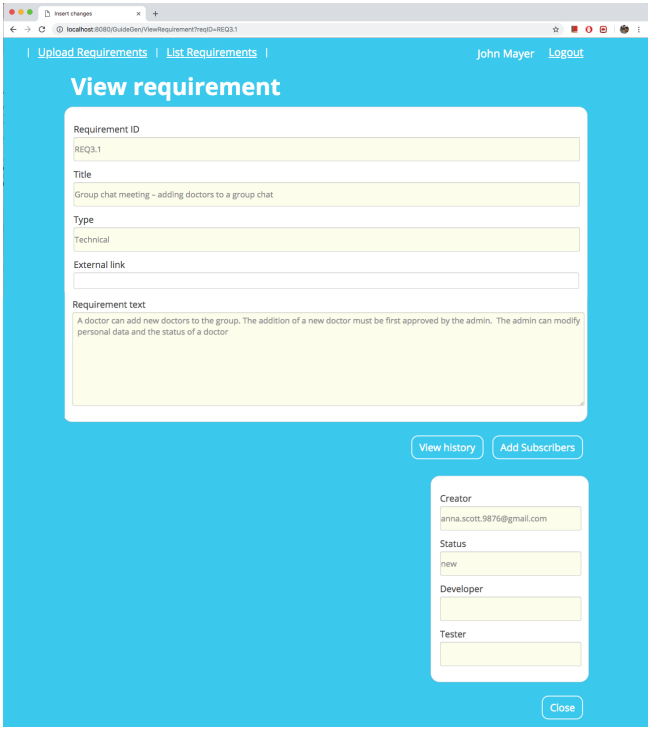


Figure 6.18: The form with details of a requirement from the perspective of a test engineer.

Action 8. Notifying subscribers about changes in requirements. When the generated suggestions are listed, the requirements engineer who made the changes to the requirement decides which of the suggestions are relevant and should be sent to the subscribed parties and which ones are irrelevant and should be ignored.

Each suggestion can be ignored by clicking on the check box “Ignore”

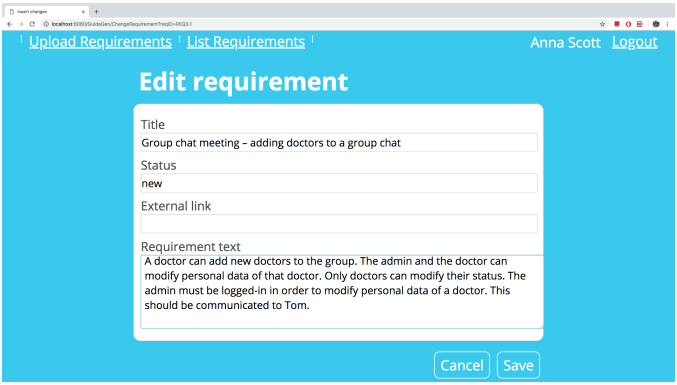


Figure 6.19: The form for editing a requirement.

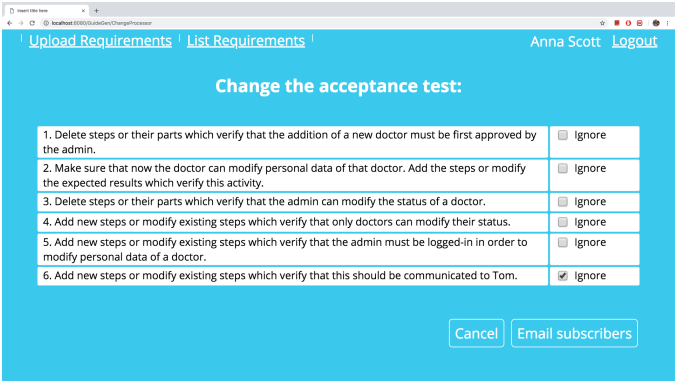


Figure 6.20: Suggestions generated for the changes in REQ3.1.

(see Figure 6.20). When the requirements engineer clicks on the button "Email subscribers", the relevant suggestions are sent to test engineers and all other subscribers via e-mail.

The e-mail contains the data about the affected acceptance test, sug-

gestions on how to adapt the test, the previous and the updated version of the changed requirement and the summarized changes (added, deleted and modified sentences), as presented in Figure 6.21.

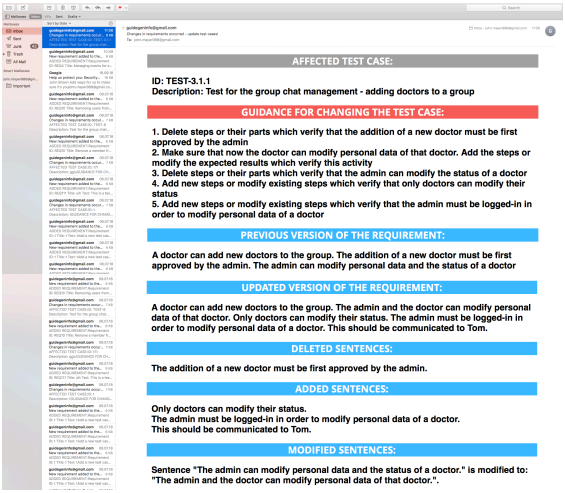


Figure 6.21: The email sent to the subscribed parties.

In addition to e-mails, GuideGen generates warnings which indicate that the related acceptance tests are no longer aligned with their requirements. The warnings are shown next to the affected acceptance tests in the list of requirements and their tests, as presented in Figure 6.22.

Not only that the warning sign is shown next to the ID of the affected test, but the ID itself also changes the color to orange when the test is non-aligned with its requirement. When a requirement has more than one related acceptance test, currently the warning sign is shown next

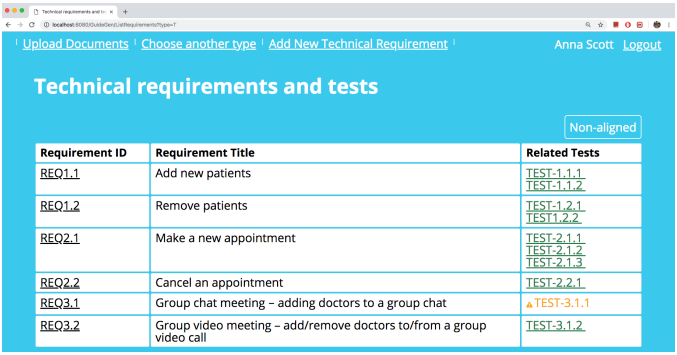


Figure 6.22: The warning sign for the non-aligned test TEST-3.1.

to each related acceptance test ID and they all change their color to orange. After integrating one of the existing methods for identifying the concrete requirement affected by the change, we will adapt this functionality, so that the warning and the orange color are applied only to the affected acceptance test.

The warning sign serves as a reminder for test engineers to adapt the non-aligned tests with their changed requirements. Figure 6.22 shows the warning from the perspective of requirements engineers and, therefore, the warning is just a text and not a link. From a perspective of a test engineer the warning is clickable, as explained in further text (See Action 9. Editing the affected test case).

Further, by clicking on the button “Non-aligned”, a user can filter all the tests that currently are not aligned with their requirements. The resulting list shows all non-aligned requirements and acceptance tests, including the newly added requirements for which the acceptance

tests yet have to be added. This feature is especially helpful for large scale systems with many requirements.

Action 9. Editing the affected test case. When a test engineer clicks on the ID of a non-aligned acceptance test, the form for editing the test is shown. The form shows the data related to the test case, such as description, execution status, steps and expected results, as well as the relevant guidance on how to adapt the test. In such a way, test engineers can easily refer to the guidance when applying changes to the affected acceptance test, as shown in Figure 6.23.

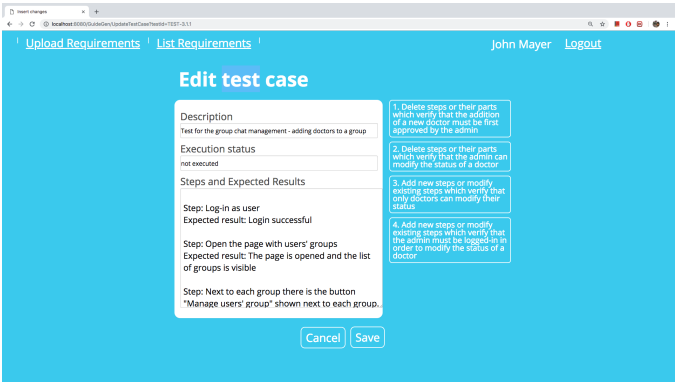


Figure 6.23: The form for editing the non-aligned test.

When there is more than one acceptance test related to a requirement, the same guidance is shown next to each of them. In our future work we plan to generate tailored suggestions for each of the involved tests.

Action 10. Notifying subscribers about changes in acceptance tests. As soon as a test engineer applies the changes to the affected accep-

tance test and saves them, a notification e-mail is sent to the requirements engineer who applied the changes to the requirement and to all other subscribers who are interested in this change in the acceptance test.

The e-mail contains the ID and the description of the test that is being updated and a notification message which indicates that the test is now aligned to its requirement, contained in the subject of the email. The content of the generated e-mail is shown in Figure 6.24.

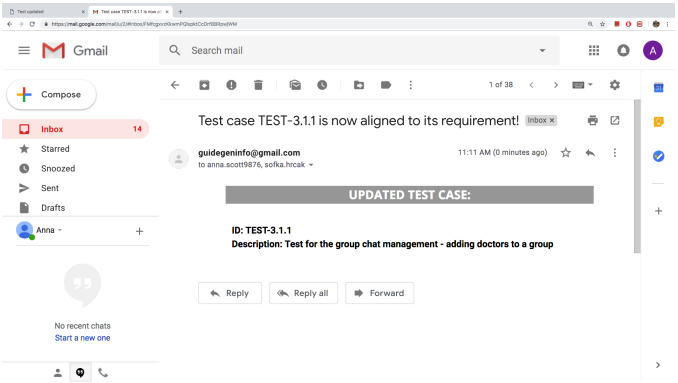


Figure 6.24: The e-mail generated when the previously non-aligned acceptance test is updated.

Action 11. Tracking the history of changes for a requirement. By clicking on the button "View history" in the form that shows requirements details (See Figure 6.17), the history of changes for that requirement is shown to the user. The history is stored automatically whenever a requirement is changed. Figure 6.25 shows the history of changes for the requirement REQ3.1.

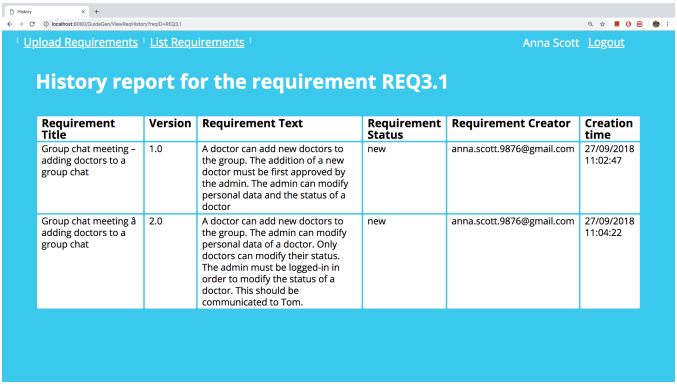


Figure 6.25: The history of changes for the requirement REQ3.1

Action 12. Adding requirement subscribers. When creating a requirement, a user can insert e-mails of subscribers in the form (See Figure 6.13), who will be notified whenever a change is applied to the requirement.

A user can add additional requirement subscribers to an existing requirement by clicking on the button “Add subscribers” in the form for viewing details of a requirement, shown in Figure 6.17 and Figure 6.18. The form for adding subscribers is shown in Figure 6.26. The user can either subscribe himself by checking the field “Subscribe me” or subscribe other users by inserting their email addresses separated by commas.

Action 13. View the details of an acceptance test. When a user clicks on the ID of an acceptance test in the list of requirements and their tests, shown for instance in figures 6.11, 6.12, 6.14 or 6.22, a page with the details of that acceptance test is shown to the user. Figure 6.27

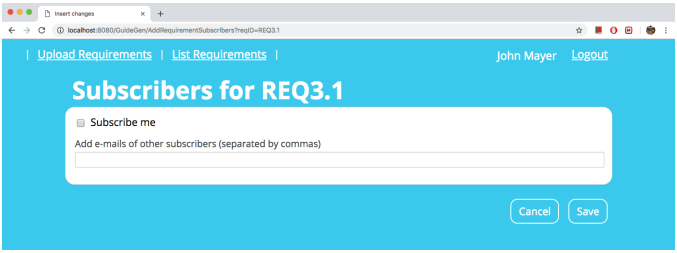


Figure 6.26: The form for adding requirement subscribers from a perspective of a test engineer.

shows the page from a perspective of requirements engineers (there is no button which enable editing), while Figure 6.28 shows how the page is seen by test engineers.

Regardless of the role of the logged user, the page shows the ID, description and steps and expected results of the chosen acceptance test, as well as its meta-data, such as the creator this acceptance test, the execution status of the test (e.g. not executed, passed, failed) and the developer responsible for the implementation of the functionality which is tested with this acceptance test. Moreover, all users can see the history of changes or add subscribers who should be notified when changes are applied to the acceptance test.

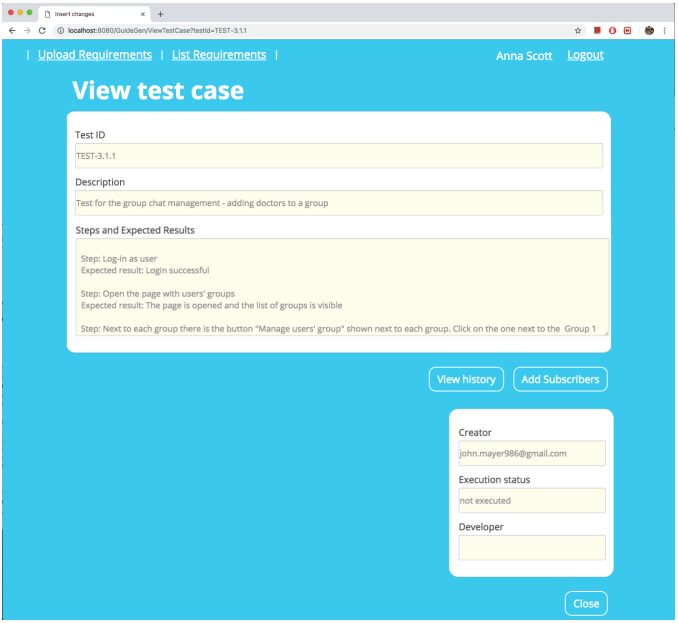


Figure 6.27: The details of the acceptance test TEST-3.1.1 from a requirements engineer perspective.

Action 14. Tracking the history of changes for an acceptance test. By clicking on the button "View history" in the form that shows test details (See Figure 6.27), the user is provided with the history of changes for that acceptance test. Similarly to the history of changes for requirements, the history of acceptance tests is stored automatically on every acceptance test edit. Figure 6.29 shows the history of changes for the acceptance test TEST-3.1.1.

Action 15. Adding acceptance test subscribers. When a test engineer creates a new test case for a requirement, GuideGen automatically sub-

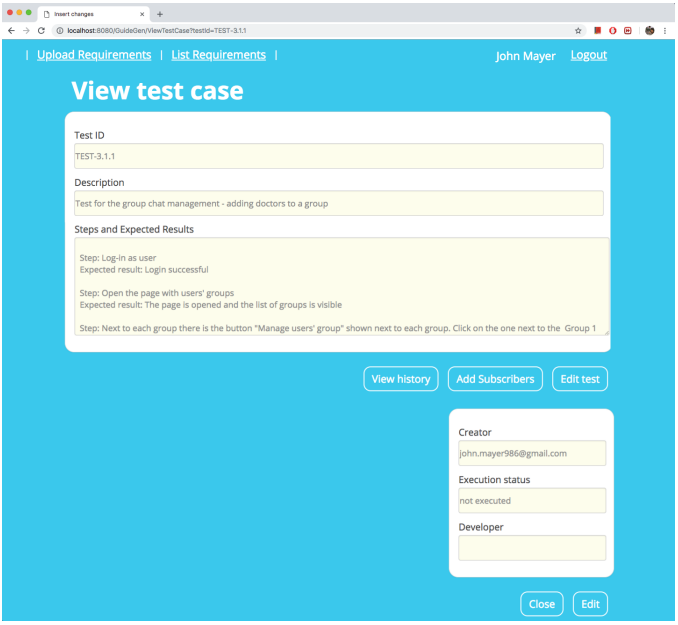


Figure 6.28: The details of the acceptance test TEST-3.1.1 from a test engineer perspective.

scribes the creator of the requirement to the newly added acceptance test. The user can add other subscribers when creating a test case by inserting their e-mail addresses in the form for adding an acceptance test, shown previously in Figure 6.16.

In order to add new acceptance test subscribers for an existing acceptance test, the user can click on the button “Add subscribers” in the form for viewing the details of the acceptance test, as shown in Figure 6.27 and Figure 6.28. The form for adding acceptance test subscribers

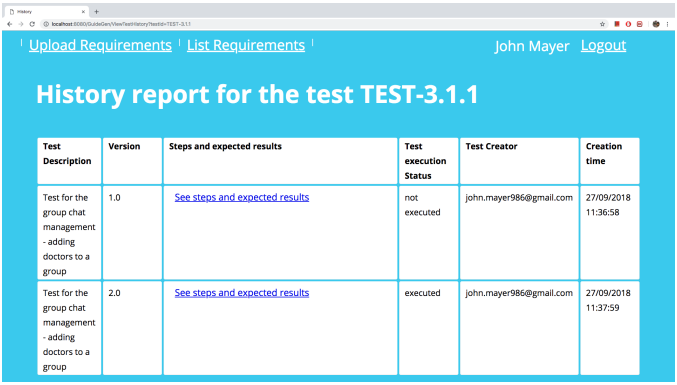


Figure 6.29: The history of changes for the acceptance test TEST-3.1.1.

looks the same as the form for adding requirement subscribers, shown previously in Figure 6.26.

6.4.3 The Architecture of GuideGen

GuideGen is a dynamic web project that follows the Model-View-Controller (MVC) architecture, as presented in Figure 6.30. According to the MVC pattern, the Model manages the data in our application, the View is a presentation layer and all background logic is managed by the Controller. In the remainder of this section we briefly describe each of these components.

The Model consists of eight java classes as shown in Figure 6.30. These classes interact one to another, so that, for instance, one requirement

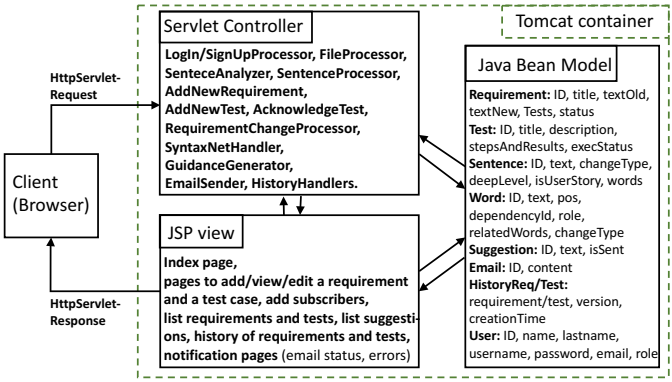


Figure 6.30: The architecture of GuideGen.

contains a list of acceptance tests, a sentence contains a list of words, etc.

The View consists of 27 JavaServer Pages (JSP) [jsp], which present the content of GuideGen to users and enable managing requirements, acceptance tests and suggestions. The majority of the pagers are shown in Section 6.4. The remaining pages are notification pages that contain information messages, such as pages about errors in the application (lost connection, problems with the data, etc.).

The Controller is the central component of our tool. It contains servlets [ser] which are responsible for all the logic in the application, such as managing users' authentication (LogIn and SignUp processor), uploading requirements and acceptance tests (FileProcessor), analyzing the changes in requirements, i.e. what sentences and what words have been changed and how (SentenceProcessor, SentenceAnalyzer, SyntaxNetHandler, RequirementChangeProcessor), gener-

ating guidance (GuidanceGenerator), notifying subscribed parties (EmailSender), managing history of changes in requirements and acceptance tests (HistoryHandlers) and managing an addition of new documents (AddNewRequirement/Test processors).

6.4.4 Availability of the Tool

We deployed GuideGen by using a Docker container. The steps for running Guidegen are described in our Git repository: `https://github.com/hotomski/guidegen`. In addition, we provide a document with example requirements and their tests. This document can be uploaded to the system or used as a template for creating a new document with requirements and their acceptance tests to be uploaded.

6.4.5 Different Versions of GuideGen

So far we have presented the GuideGen tool in its current version. When developing GuideGen, we have followed an *iterative* research path with three development iterations and two evaluations so far. It is therefore important to know which versions of GuideGen were used in our evaluations. We provide this information in Table 6.5.

All features of GuideGen which are not listed in Table 6.5 (for example, the basic feature of generating guidance when a requirement is changed) are common for all versions of GuideGen. The first version of GuideGen that we evaluated in 2016 (see Sect. 6.5 below) did not

Table 6.5: The differences between the current and previously evaluated versions of GuideGen.

Features	Evaluation I [HBCG18]	Evaluation II [HG18b]	Current
Generating guidance for changes of numerical values	✗	✓	✓
Showing guidance in the form for editing a test case	✗	✓	✓
Showing warnings in the application for non-aligned documents	✗	✓	✓
Sending e-mails and showing warnings for newly added requirement	✗	✓	✓
Sending e-mails when acceptance tests are adapted according to the requirement changes	✗	✓	✓
One-to-many relationship between a requirement and its tests	✗	✗	✓
Possibility to see the outcome of a test execution	✗	✗	✓
Possibility to see the status of a requirement	✗	✗	✓
Showing the meta-data of a requirement or an acceptance test	✗	✗	✓
Tracking the history of changes in requirements and acceptance tests	✗	✗	✓
Personalizing the users (logging-in, signing up and account privileges)	✗	✗	✓
Differentiating between business and technical documents (kept separately)	✗	✗	✓

have any of the features listed in Table 6.5. Based on the results of the first evaluation, we added the top five features listed in Table 6.5

to GuideGen. That version was used in the second evaluation (see Sect. 6.6). The remaining new features have been added to GuideGen recently as a consequence of the lessons learned in the second evaluation.

6.5 Quantitative Evaluation and the First Qualitative Assessment

In this section we describe our first evaluation of GuideGen. The main goal of this evaluation was to obtain quantitative information about the correctness, completeness, understandability and relevance of the suggestions generated by GuideGen when applied to requirements changes in real industrial projects.

This evaluation was conducted between September and December 2016, using an earlier version of GuideGen, as described in the previous section.

6.5.1 Study Design

We first contacted companies and asked them whether they could provide us real data about requirements, associated acceptance tests and changes to the requirements. We managed to find three companies who were willing to give us the data and also committed to assess the guidance produced by GuideGen.

After receiving the data sets from the three companies, we first pruned the data as follows: we omitted all requirements that had not been changed at all or did not have acceptance tests associated with them and we removed irrelevant changes such as added or deleted punctuation marks, spaces or empty lines. We then ran our tool in order to generate the guidance that was further evaluated. Each of the generated suggestions in the guidance was evaluated by experts from the three companies. Finally, we showed the tool prototype to the experts and conducted a short qualitative assessment of GuideGen.

Companies overview We evaluated GuideGen by applying it to real-world data sets with requirements changes provided by three companies. An overview of the companies is provided in Table 6.6.

Table 6.6: Characteristics of the companies that provided us data sets from one of their projects.

Company	C1	C2	C3
Domain of activity	Access control and security solutions	IT integration, cloud services	Automation for warehouses and distribution centers
Software process model	Agile(Scrum)	Agile (Scrum)	Waterfall
# of employees in total	16000	500	2500
# of employees on the project	120	100	500
Country	Switzerland	Serbia/ Germany	Switzerland

Participants The generated guidance was assessed by seven experts from the three companies. An overview of the experts and their experience is provided in Table 6.7.

Table 6.7: Characteristics of the experts who participated in the study.

Company	Participant	The role of participant	Years of experience in IT	Years on the current position
C1	P1	Requirements engineer	10	4
C1	P2	Senior test analyst	12	4
C2	P3	Requirements engineer	6	3
C2	P4	Senior test engineer	7	4
C3	P5	Technical business analyst	10	5
C3	P6	QA manager	12	6
C3	P7	Test engineer	4	4

Table 6.7 presents the roles of the participants exactly the way the participants reported them during the interview. Although the table contains different roles, we can divide them in two main groups: (1) requirements engineers and (2) test engineers. For instance, a technical business analyst performs the same tasks as the other two requirements engineers. The typical tasks of the practitioners with the requirements engineer role are to:

- write and maintain requirements documents,
- communicate to clients that are outside the organization with a goal to elicit requirements, clarify uncertainties or present the work progress and

- communicate to the rest of the team, including testers, in order to clarify how requirements should be implemented or tested or to communicate the changes and resolve ambiguities.

Similarly, the test analyst, two test engineers and the QA manager, all have the same duties and perform similar tasks. They all write and maintain test documents and perform testing activities, such as executing manual acceptance tests and reporting and maintaining issues in issue trackers. The differences are in the years of experience or whether these roles have additional duties, such as managing the team of testers as, for instance, P6 does.

Data collection and analysis. For our evaluation, we needed data records containing the old and the changed version of a requirement and the associated acceptance tests. Table 6.8 characterizes the data sets.

Table 6.8: Characteristics of the data sets used in our evaluation study.

Company/ Data set	Type of requirements	# of requirements in the data set	# of considered requirements	# of evaluated changes
C1/DS1	User story	157	20	28
C2/DS2	User story	30	30	37
C3/DS3	Traditional*	5301	398	197

Pruning the data sets We pruned the received data sets as follows: (1) we omitted all requirements that have not been changed at all or did not have acceptance tests associated and (2) we omitted irrelevant changes, such as added or deleted punctuation marks, spaces or empty lines. The pruning yielded a total of 448 changed requirements. Our

tool filters out semantically irrelevant changes such as addition or deletion of determiners or corrections of typos. On the other hand, for several requirements there was more than one change. So we eventually could evaluate a total of 262 changes (28 for C1, 37 for C2 and 197 for C3).

Running the tool. For every of the 262 evaluated changes, we generated guidance for how to change the associated acceptance tests using our tool prototype. We uploaded the old version of the requirements into the tool, replaced each of them with the new version, and recorded the generated guidance.

Assessing the generated guidance. 95 changes were fully assessed by two or three experts. We created a questionnaire[4] in which, for every requirement, we presented the old and the changed requirement, the associated acceptance tests and the guidance for changing the acceptance tests generated by our tool.

For each suggestion provided in the guidance, we asked six questions to assess the quality of the suggestion: (1) Is the suggestion correct in terms of actions that need to be performed? (2) Is it grammatically correct? (3) Is it complete? (4) Does the expert understand what has been suggested by the tool? (5) Would the expert be able to perform an update of the impacted acceptance test without any further clarifications? (6) Is the suggestion redundant or unnecessary? Finally, we asked whether there is anything missing from the guidance for a changed requirement (i.e., from the set of all suggestions generated

[4] https://goo.gl/B2GPu3 For confidentiality reasons, the file does not contain the real data from our data sets, but only the example shown in this paper.

for that requirement). Questions 1-3 and 5 had to be answered on a five-point Likert scale (from “strongly disagree” to “strongly agree”). In case of non-agreement, the expert was asked to provide an explaining text. Every answer other than 5 (‘strongly agree”) is interpreted as a non-agreement. Question 4 was a yes/no question, while Question 6 and the final question about missing suggestions were answered as free text.

In company C3, due to limited availability of the experts, only 30 suggestions could be thoroughly assessed by all three experts. The suggestions generated for the remaining 167 changes could only be assessed for correctness by a single expert.

When performing this evaluation, GuideGen did not consider changes of numerical values as relevant. The approach is later extended, since the results showed that changes of numerical values in requirements affect the related acceptance tests.

First qualitative assessment Although the study focused on the quantitative evaluation in terms of the correctness and completeness of the generated guidance, we additionally performed a short qualitative assessment of the approach and the tool. When the experts had finished answering the questionnaire for all changed requirements assigned to them, we showed them the tool and conducted a short interview about the usefulness and applicability of GuideGen[5]. The practitioners gave us some useful advice on how to adapt GuideGen in order to improve its usefulness and applicability.

[5]https://goo.gl/LZtQWg

When interviewing the participants, we took notes and recorded the interviews. As the interview instrument was rather short and we had only seven interviewees, we did not formally transcribe the interviews. Instead, we compared our notes with the recordings, corrected errors and omissions, and then used the updated notes as the basis for our analysis.

6.5.2 Results

In this sub-section we present the results of the assessment of the generated guidance by the experts and some key insights from the follow-up interviews.

All 262 changes were correctly identified in terms of the change type, showing that the algorithm for identifying added, deleted and modified sentences with a modification threshold of 0.6 performs accurately. Table 6.9 presents the results of the evaluation of the guidance generated for 95 changes in requirements by the experts.

For calculating the percentages in Table 6.9 for the questions answered on a Likert scale, we interpreted the values 4 (“Agree”) and 5 (“Strongly agree”) as “yes”. Analogously, we interpreted 1 (“Strongly disagree”) and 2 (“Disagree”) as “no”. 3 (“Neutral”) was interpreted according to the textual explanation provided by the experts. From eleven such answers three were interpreted as “yes” and eight as “no”.

Table 6.9shows that in C1 and C2 the experts assessed more than 80% of the suggestions as correct in terms of actions. In C3 one expert was

Table 6.9: The quality of the generated suggestions based on an assessment by industrial experts.

Generated in total/ assessed	Company/ Participant	Correct in terms of actions	Grammatically correct	Complete	Understandable	Self-explanatory	Redundant/ unnecessary	Missed changes
28/28	C1/P1	89.2%	82.1%	100%	100%	75%	7.1%	3.6%
	C1/P2	89.2%	82.1%	100%	100%	75%	7.1%	3.6%
37/37	C2/P3	81%	67.5%	94.6%	100%	75.6%	10%	5.4%
	C2/P4	81%	67.5%	94.6%	100%	75.6%	10%	5.4%
197/30	C3/P5	50%	86.6%	96.6%	93.3%	70%	50%	3.3%
	C3/P6	70%	80%	93.3%	100%	73.3%	30%	3.3%
	C3/P7	66.7%	86.6%	96.6%	93.3%	73.3%	33.3%	3.3%

more negative than the other two, especially regarding the correctness in terms of actions. This is due to a misunderstanding: expert P5 classified all redundant suggestions as wrong in terms of actions, i.e., when they were actually correct, but unnecessary. Since P7 found 66.7% and P6 70% of the suggestions to be correct in terms of actions, we can consider the correctness of our guidance for data set 3 to be at least 66.7%.

The following text contains a change (in the acceptance criteria of a user story) for which GuideGen does not work such well:

> "- The section 3 contains:
> -- Doctors' corner
> -- Register your practice opens a form inline or a popup with:
> --- Name of your practice (mandatory)
> --- Contact phone (mandatory)
> --- Contact e-mail (mandatory)
> --- Give us your contact details and we will get back to you soon!"

According to the experts, the text means that Section 3 of a web page contains a label "Doctors' corner" and a button "Register your practice". When a user clicks on the button, an inline form or a pop-up window is displayed. The change in the requirement is that an additional message shall be displayed in this window.

For this change, the GuideGen tool generated the following suggestion, which the experts considered to be wrong both in terms of actions and grammatically: "Add new steps or modify existing steps which verify that the section 3 contains register your practice opens a form inline or a pop-up with give us your contact details and we will get back to you

soon!". This result may indicate that our approach does not perform well on ill-structured texts. However, it may also indicate that our treatment of enumerations (cf. sentence level analysis in Sect. 6.3.1) needs improvement.

The last column in Table 6.9 presents the number of changes that were relevant, but not detected by GuideGen. In C1 a noun with no related verbs was added. This was classified as an irrelevant change and hence no guidance was generated. Further, as previously explained, as numerical values were not considered as relevant, no guidance was generated for two such cases in C2 and one in C3. This problem is fixed in the next release of the tool.

As stated above, the guidance for 167 changes in requirements from company C3 could not be evaluated fully due to limited availability of the experts. Table 6.10 shows the results of the assessment of the generated suggestions for these changes.

We found that 70.6% were correct in terms of actions, while 24% were incorrect because the changes only rephrased a requirement or added or deleted only clarifications or notes. A small percentage (5.4%) of wrong suggestions were due to limitations of our prototype tool (e.g., wrongly identified dependencies).

Next, we present the main findings from the follow-up interviews with the experts regarding the overall usability and usefulness of GuideGen. All experts stated that GuideGen can be helpful in communicating changes on time and with less effort, it can help test engineers to make a decision on how to update acceptance tests and they would be

Table 6.10: Suggestions assessed for correctness in terms of actions by a single expert only.

Company/Participant(role)	C3/P6(QA)
Assessed suggestions	167
Correct in terms of actions	70.6%
Wrong due to rephrasing only	10.2%
Wrong as only clarifications or notes are added/deleted	13.8%
Wrong due to tool limitations	5.4%

willing to slightly adapt their style of writing requirements in order to ensure better quality of guidance. Four experts emphasized that one of the reasons for wrongly generated guidance was the poor quality of the requirements. They stated that suggestions can be too general, but that this is directly related to the level of detail specified in the requirements.

The experts from C1 stated that the approach would be even more useful if it could show the suggestions directly in the acceptance test document. With respect to the usability of the tool, P1 and P2 suggested an improvement of the user interface so that the tool (1) navigates directly to the steps that are suggested to be changed (2) shows warnings about the non-aligned documents somewhere in the application.

Participants P1 and P4 suggested that not only changes in requirements should be communicated via e-mails, but also changes in affected acceptance tests.

6.5.3 Threats to Validity

Internal and construct validity. Our quantitative evaluation strongly depends on the expertise of the people who assessed the guidance generated by GuideGen. In order to foster validity, we aimed at assessing each guidance by at least two experts. In company C3, due to limited availability of experts, we could assess only 30 cases this way, while the rest was evaluated only in terms of correctness by a single expert. We tried to mitigate this problem by including all types of changes in the fully evaluated sample from company C3. Even with this restriction, the workload for the experts was high, since they needed to answer six questions per 28 and more suggestions, which might impact the quality of their answers. Therefore, we provided an online access to the questionnaire, so that the experts could answer the questions in iterations.

External validity. The generalizability of our results is limited by the fact that our evaluation covers data sets from only three companies. We tried to improve generalizability by including both agile and traditional requirements artifacts as well as different types of changes in our data sets. Although the study involves only seven participants, we had at least two participants per data set and we tried to keep diversity in terms of roles, so that requirements engineers and test managers are included.

6.6 Qualitative Evaluation

After the first evaluation, we adapted GuideGen, based on the comments and suggestions received from the practitioners. We then performed a second evaluation study in which we aimed at evaluating GuideGen in terms of its perceived usefulness for practitioners and its applicability to real software projects.

The study was conducted from October 2017 to January 2018. The version of GuideGen that was used in this study differs from the current version of GuideGen as described in Section 6.4.5.

In this section, we briefly describe the study and summarize the main results, while the details are described in [HG18b].

6.6.1 Study Design

We conducted a *qualitative experiment* [Kit96] with twenty-three practitioners from ten companies based in Europe. To make this paper more self-contained we provide an overview of the companies in Table 6.11. From these ten companies we interviewed twenty-three requirements and test(QA) engineers with at least five years of experience. For more details about the participants and their teams please refer to [HG18b].

The experiment consists of three parts: an introductory presentation, a tool trial and an interview.

Table 6.11: An overview of the companies

Company	Domain of activities	Country	# of employees
C1	Power industry	Serbia (International)	1000 (10000 worldwide)
C2	Access-control solutions	Switzerland (International)	1000 (10000 worldwide)
C3	Distribution Solutions	Switzerland (International)	500 (2500 worldwide)
C4	Enterprise Content Management	Switzerland (International)	50 (7000 worldwide)
C5	IT project and product management	Serbia (Germany, India)	20 (500 in total)
C6	Online marketing and online shopping	Germany (International)	1200 (3000 worldwide)
C7	Content Management Systems	Netherlands (International)	60 (600 worldwide)
C8	Software testing services	Serbia (Austria, Switzerland)	120 (500 in total)
C9	Data and Analytics	Denmark (Sweden)	250 (1000 in total)
C10	Website and mobile development	Serbia (Germany)	110 (350 in total)

1. Introductory presentation. The first author explained the GuideGen approach in a 20 minute talk to a group of business and technical practitioners. Then she asked for requirements and test engineers who were interested to try out GuideGen and evaluate it. We did this to ensure that only people who are interested and knowledgeable in the field of requirements engineering or testing participated in the experiment.

2. Tool trial. Although the participants already got familiar with the GuideGen tool by attending the presentation, we encouraged them to try it themselves, using examples from their own projects. The tool trial lasted between 30 and 45 minutes.

3. Interview. After a five minute break, we conducted semi-structured interviews with the participants. Each interview consisted of two main parts: an exploratory part referring to the current issues and challenges faced when requirements change and an evaluation part about the usability and applicability of GuideGen. The interviews lasted between sixty and ninety minutes.

6.6.2 Results

We classified our results into four groups, based on the research questions we defined in this study:

(1) Current issues with changing requirements;

(2) Benefits and strengths of GuideGen that can mitigate some of the identified issues;

(3) Applicability of GuideGen to real industrial projects;

(4) Usability of GuideGen as perceived by practitioners.

The most frequently reported issues with changing requirements can be classified into communication, documentation and tool issues. For instance, the communication issues reported are late- or non-communicated changes to test engineers, passing incomplete information from requirements to test engineers and inability to track changes due to different communication channels. Documentation issues are outdated documents, incorrect acceptance tests and bad quality of requirements documents. With respect to the issues with the currently used tools, we found that the most challenging issues are usage of too many different tools for document management, complexity of the tools and inconvenient notification system.

The practitioners recognized the following benefits of GuideGen:

B1. With GuideGen the communication of changes can be faster;

B2. Requirements changes would be no longer be missed or overseen by testers;

B3. Acceptance tests would be updated faster and easier;

B4. With GuideGen requirements and acceptance tests would be kept closely together;

B5. GuideGen is simple to use;

B6. GuideGen would serve as a reminder for updating documents.

More details about each of the benefits are described in [HG18b].

With regard to the applicability of the GuideGen tool, the practitioners found the tool to be applicable for smaller projects as it is, while for bigger projects GuideGen should either be incorporated as a plug-in into existing document and project management tools, such as Jira or TFS, or it would have to be extended with numerous features. The most frequently reported features that are perceived as relevant, but are missing in GuideGen, are summarized in Table 6.12, which was earlier presented in [HG18b]. In the meantime, we have implemented all these features in the GuideGen tool (see Table 6.5).

Table 6.12: The missing features reported by at least two participants in two different interviews

Missing features	Participants
A possibility to see the outcome of the test execution (e.g. skipped, passed, failed, not started...)	P2, P4, P6, P12, P15, P19, P21
A possibility to see the status of a requirement (e.g. new, assigned, opened, finished...)	P6, P12, P13, P15, P17, P21
Tracking the history of changes in requirements and tests	P5, P7, P16, P20, P23
Personalize the user (having different accounts with different privileges)	P7, P9, P14, P16, P21
Different structure for keeping the documents (more folders for grouping different types of requirements, different projects or features)	P3, P5, P6, P12

Besides missing features, the participants mentioned two further reasons why they would not replace their current tools for managing requirements and acceptance tests by GuideGen. The first reason is the complexity and cost of data migration and configuration setup, and the second is the effort needed for adjusting to new tools and learning how to use them.

Finally, the fact that GuideGen (in the version used in this evaluation) was limited to only one acceptance test per requirement was considered as a major shortcoming by the participants.

Despite these limitations, GudeGen is perceived as useful or extremely useful tool by the majority of the participants. For more concrete results of the quantitative assessment of usefulness of GuideGen please refer to [HG18b].

6.6.3 Threats to Validity

Internal and construct validity. In order to decrease the threats to internal validity, we carefully designed our interview questions, discussed them with a group of researchers, improved them and performed a pilot interview, which limited the reliability threat. However, reliability threats regarding researcher bias cannot be completely ruled out, because the interviews were conducted by only one researcher. In order to avoid selection bias, we avoided personal contacts when we performed the interviews for the study.

The main threat to construct validity is that the participants used GuideGen only for less than one hour. To limit this threat, we designed

our study such that the participants had the opportunity to try out the GuideGen tool themselves, using examples from their own work, so that they are able to provide a rational evaluation.

Although the participants were told to be objective and express all their concerns towards the tool, we cannot completely reject the possibility of politeness bias caused by the fact that a co-creator of GuideGen presented the tool and conducted the interviews.

External validity In order to limit the threats to external validity, we tried to keep diversity in terms of structure of the chosen companies, their size, location, domain of responsibilities, processes applied within the companies and the way that requirements and tests are documented.

6.7 An assessment of the existing approaches for change impact analysis

Although one-to-one relationships between requirements and acceptance tests frequently occur in practice [HBCG16], we learned in our second evaluation that GuideGen should support more than one acceptance test per requirement and produce tailored guidance for every impacted acceptance test when a requirement is changed (see Section 6.8.3 and Section 6.8 (*A one-to-one relationship between requirements and their acceptance tests is not sufficient*)). For achieving this, GuideGen will have to analyze which change in a requirement

impacts which of the associated acceptance tests. As a first step towards implementing such an analysis in GuideGen, we experimentally assessed how well existing approaches for change impact analysis between textual documents perform when applied to requirements and acceptance tests. For this assessment we used a small data set from industry and two existing systems for change impact analysis: NARCIA [ASG+15a] and ImpRec [BGW13], [BWRR17]. In the remainder of this section we describe the experiment design, present the results and discuss the applicability of the existing approaches and their possible integration with GuideGen.

6.7.1 Experiment Design

The choice of tools for the experiment. In order to assess how well the existing techniques for change impact analysis perform when applied between requirements and their acceptance tests, we firstly investigated the available tools that can be used without adaptations of their functionality. We investigated tools that are intended to work with unstructured textual artifacts, excluding, for instance, the ones that calculate impact between source code and other artifacts, such [CKG12] or [DWW+14]. In addition, tools should be publicly available. Both NARCIA and ImpRec satisfied our conditions and were, therefore, chosen for the experiment.

The data set used in the experiment. For this experiment, we used a data set obtained from the company C2 (please refer to Table 6.6 for the details about the company C2). This is a new data set which

has not been used in any of our previous studies. The data set was collected in June 2018.

In order to calculate the impact that changes of a requirement have on the related acceptance tests, we used requirements that have more than one acceptance test related to it and have been changed at least once. In addition, we needed to have both the old and new versions of the requirement available, so that we could identify the exact changes. Considering these conditions, we found nine such requirements and twenty-three acceptance tests related to them. One of the requirements has five, one has four and the remaining ones have two associated acceptance tests. Two requirements have been changed two times; the others were changed once. This resulted in eleven revisions that we could use in our experiment.

The experiment setup The experiment was conducted in October 2018. In order to define the ground truth for the experiment, we needed to know which test cases were adapted based on the changes in the requirements. Since the data set was small, we analyzed this manually together with the test engineer from company C2, who performed the adaptations. The analysis resulted in knowing how the requirement is changed and which test case was actually affected by that change.

We then loaded the data into the tools, applied the changes and analyzed the impact obtained from the changes. For every revision of a requirement, we calculated the impact score between the requirement and its related acceptance tests. Since GuideGen is designed such that it knows which acceptance tests are associated to a requirement,

we only calculated the impact scores between a requirement and its associated tests, rather than between all requirements and tests in the system. The acceptance test with the highest score is considered to be the one that is the most affected. We checked whether the found test is actually the one that the test engineer declared as impacted.

Below we describe how we used NARCIA and ImpRec to calculate the impact scores.

1. Working with NARCIA: NARCIA is the tool that calculates impact on other requirements when a requirement is changed. Therefore, the original input for the tool is a list of all requirements present in the system. In order to calculate the impact between a requirement and their acceptance tests, our input for NARCIA was a requirement and the acceptance tests related to the requirement.

As soon as changes are applied to a requirement and saved, NARCIA analyzes what has been changed and summarizes the changes. Then the user specifies the propagation conditions that are going to be used in the change impact analysis. The propagation conditions specify how a change should propagate in acceptance test documents (originally in other requirements documents). The conditions are words and phrases, which are then considered when calculating the impact. In Table 6.13, we summarize the propagation conditions that we specified for calculating impact based on the changes from our data sets. We defined the conditions in such a way that they can be later automatically defined by GuideGen based on the SyntaxNet analysis.

For some change types we defined several propagation conditions, as shown in Table 6.13. For instance, when a sentence is added, in order

Table 6.13: The propagation conditions used based on the types of changes. *Previous sentence is a sentence that appears before the changed one in the requirement.

Change type (number of occurrence)	Whole sentence	Changed and relevant words from the modified sentence	Changed and relevant words from the previous* sentence	Changed and relevant words from the modified and the previous sentence	Deleted words only	Main part of the sentence + changed (enumerated) part
A sentence is added (4x)	✓	✓	✓	✓		
A sentence is deleted (2x)	✓	✓				
A conjunction verb is added (2x)		✓				
A conjunction noun is added (1x)		✓				
A verb and a noun are deleted (1x)		✓			✓	
An item in an enumerated sentence is deleted (1x)						✓

to calculate the impact of that change to the related acceptance tests, the following propagation conditions are specified:

(1) the whole sentence that has been added,

(2) only relevant words that have been added,

(3) only relevant words from the previous sentence,

(4) relevant words from the added and from the previous sentence.

We illustrate this with a concrete example taken from the data set used in the study. The following change has been applied to requirement 3:

> Assign access group. As a site operator, I want to assign access groups to a person, so that I don't have to repeatedly assign access zones and time profiles to every person. Groups can be modified by adding or deleting persons. Groups can also be entirely deleted.

For this change we generated the following propagation conditions as query parameters:

(1) Groups can also be entirely deleted (the whole sentence),

(2) Groups AND can be AND deleted (only relevant words from that sentence),

(3) Groups AND can be AND modified AND adding AND deleting AND persons (only relevant words from the previous sentence),

(4) Groups AND can be AND modified AND adding AND deleting AND persons AND Groups AND can be AND deleted (relevant words from the previous and changed sentence).

For each of the specified propagation conditions we calculated the impact scores. In this example this resulted in four comparisons of calculated impact scores. In total, we made twenty-five impact comparisons based on change types applied to requirements and different propagation conditions.

2. Working with ImpRec: ImpRec performs two main tasks: 1. It calculates similarity between the issue reports based on the query parameters; 2. It recommends the artifacts potentially impacted by resolving incoming issue reports. For recommending such artifacts, ImpRec uses the knowledge database built based on the previously calculated impacts.

Since we do not have previous impacts calculated, we used ImpRec for calculating the similarities between a requirement and its acceptance tests based on the changes that are applied to a requirement. The same propagation conditions, presented in Table 6.13, were used as query parameters for calculating similarities. This resulted in twenty-five similarity calculations for the applied change types and defined query parameters.

6.7.2 Experiment Results

In this subsection, we present the results both for NARCIA and ImpRec. In general, the results are encouraging. We also found that that

the choice of the propagation conditions/query parameters strongly affects the quality of the results.

Results obtained by using NARCIA. By choosing the best propagation conditions, NARCIA was able to correctly identify the affected acceptance test in ten out of eleven revisions (90.1%), and in the worst case, when considering only the badly chosen conditions, NARCIA correctly detected test cases from seven revisions (63.6%). Table 6.14 shows which propagation conditions gave the correct results for different types of changes applied within the data set. Providing correct results means that the acceptance test with the highest score among other related tests is exactly the one declared by the test engineer.

In only one case none of the propagation conditions was satisfying (the addition of a sentence in the requirement 8) and, consequentially, the affected acceptance test could not be identified. This was due to a complete word mismatching. The added sentence was clarifying a detail that was not specified in any of the related acceptance test descriptions. From pure syntactic analysis it was not possible to conclude that the change actually has an impact on the affected acceptance test, but deeper understanding and domain knowledge was needed in this case. For other changes, there was at least one condition that resulted in identifying the correct acceptance test.

In our example, there were two tests related to requirement 3. Although NARCIA correctly identified Test 2 with a higher score than Test 1 for all four conditions, the third condition (C/P3 in Table 6.14) gave the biggest difference between the scores. That was the case when only relevant words from the previous sentence were specified in the propagation condition.

Table 6.14: Propagation conditions/query parameters that give the correct results.

Requi-rement	Revision	Number of tests	Change type	Propagation conditions/ query parameters applied	Propagation conditions that gave correct results when using NARCIA	Query parameters that gave correct results* when using ImpRec
1	1	2	A sentence is deleted	C/P1: the whole deleted sentence; C/P2: relevant words from the changed sentence	C/P1, C/P2	NONE
1	2	2	A conjunction verb is added	C/P1: relevant words from the changed sentence	C/P1	C/P1
2	1	4	A sentence is deleted	C/P1: the whole deleted sentence; C/P2: relevant words from the changed sentence	C/P1, C/P2	C/P1, C/P2
3	1	2	A sentence is added	C/P1: the whole added sentence; C/P2: relevant words from the added sentence; C/P3: relevant words from the previous sentence; C/P4: relevant words from the added and the previous sentence	C/P1, C/P2, C/P3, C/P4	C/P4
4	1	5	A verb and a noun are deleted	C/P1: relevant words from the added sentence; C/P2: deleted words only	C/P1, C/P2	NONE
5	1	2	A conjunction noun is added	C/P1: relevant words from the added sentence	C/P1	NONE
6	1	2	A sentence is added	C/P1: the whole added sentence; C/P2: relevant words from the added sentence; C/P3: relevant words from the previous sentence; C/P4: relevant words from the added and the previous sentence	C/P3	NONE
7	1	2	A sentence is added	C/P1: the whole added sentence; C/P2: relevant words from the added sentence; C/P3: relevant words from the previous sentence; C/P4: relevant words from the added and the previous sentence	C/P1, C/P2, C/P4	C/P3 C/P4
7	2	2	A conjunction verb is added	C/P1: relevant words from the added sentence	C/P1	C/P1, C/P2, C/P4
8	1	4	A sentence is added	C/P1: the whole added sentence; C/P2: relevant words from the added sentence; C/P3: relevant words from the previous sentence; C/P4: relevant words from the added and the previous sentence	NONE	NONE
9	1	2	An item in an enumerated sentence is deleted	C/P1: Main part of the sentence + changed(enumerated) part	C/P1	C/P1

Results obtained by using ImpRec. ImpRec could identify the correct acceptance tests in six revisions (54.5%) when the right query parameters were taken into consideration. In the worst case, when only bad parameters were chosen, ImpRec could detect the actually affected tests in only three revisions (27.3%). For the revisions 1, 6 and 8 ImpRec did not identify any of the tests to be affected at all, and for the remaining revisions the scores were equal for all related tests, and therefore, we interpreted this as incorrect.

In our example, ImpRec detected the correct acceptance test for the change of requirement 3 only when C/P4 from Table 6.14 was chosen (when the relevant words from the previous and changed sentence were specified as query parameters).

The applicability of the approaches to GuideGen The results show the importance of the conditions and parameters that are chosen when calculating impact of changes in a requirement to its acceptance tests. In GuideGen we can implement all these search criteria based on the change type and combine them to get the best results.

Since NARCIA showed rather good results and is written in Java, it is our primary candidate for the integration with GuideGen and for further investigations. In particular, we will have to address the problem of low performance when a sentence has been added to a requirement. Although ImpRec performed worse than NARCIA, it still deserves further investigation, as we only partially used its functionality in our experiment. This is due to our data set, which did not have enough information for ImpRec to calculate the change impact between documents, but it calculated only the similarity. In our future work, we

plan to investigate the performance of ImpRec on data sets that contain the history of the previous changes and the previously affected tests. Technically, the integration of ImpRec into GuideGen would be possible, although ImpRec is written in C#.

6.7.3 Threats to Validity

In order to mitigate threats to *internal validity*, the ground truth was defined by a test engineer who adapted the test cases based on changes in her or his requirements. Since the data set was small, the test engineer manually labeled all tests that were actually affected by the changes applied to the related requirements. Therefore, we could assess the performance of the tools without ambiguities. To mitigate the threat stemming from the small size of the data set, we made sure that the data set contains different types of change patterns applied to requirements.

With respect to *external validity*, i.e., the generalizability of our results, we see two major threats: we used only a small data set from one company and all the changes to the affected acceptance tests were applied by only one test engineer. We partially mitigated these threats by taking the data from a real-world, ongoing project, so that we could assess how the tools perform when applied on documents currently used in industry. As a part of future work, the experiment should repeated with a larger and more diverse data set.

6.8 Discussion

In this section we discuss the results from our two evaluation studies and the lessons learned.

6.8.1 Correctness of the Generated Suggestions

The results presented in Table 6.9 show that the quality of the generated guidance differs from company to company. This is not surprising as the outcome of our natural language processing techniques depends on the type and quality of requirements artifacts and on the content that is being changed in these artifacts.

GuideGen performs better for user stories than for traditional requirements. This is probably due to the fact that user stories typically are more concise and describe features more precisely than traditional requirements do. Further, text changes in traditional requirements documents often do not bring any novelty to the feature that is being described, but only provide clarifications or simply rephrase the text.

The complexity of a sentence also affects the quality of the guidance generated. On the one hand, very short or incomplete sentences affect both the correctness and completeness of suggestions and may even cause the omission of relevant changes. On the other hand, long, complex sentences which contain one or more relative clauses or statements in parentheses may cause problems: word classes, their

grammatical functions and dependencies between words in a sentence may be wrongly identified, which leads to wrongly generated guidance.

Our approach currently cannot recognize certain types of irrelevant changes, for example, when comments such as “This should be communicated to Tom” are added. Wrong suggestions are generated in this case. However, our tool allows a requirements engineer to remove such false positives easily before communicating changes and generated guidance to subscribers (cf. Fig. 6.21).

6.8.2 Usefulness and Applicability of GuideGen

Our qualitative evaluation confirmed that GuideGen is a useful and needed tool, because the issues caused by non-aligned or outdated documents and poor communication of changes are still present in industry. However, although there is a lack of tool support, GuideGen cannot help if the internal organizational culture does not encourage employees to keep documents aligned and consistent.

In terms of applicability, we found that the GuideGen cannot replace currently used tools, but rather should be incorporated into them. This is for three main reasons:

(1) Missing features: The participants stated that GuideGen would have to provide numerous additional features in order to be able to support all documentation management tasks.

(2) Data migration and configuration setup: In complex systems, data migration is a difficult and expensive task and, therefore, experts hesitate to start to use a new tool, which would force them to migrate all data and configurations set up in the currently used tools.

(3) Learning effort: The participants argued that learning how to use a new tool and adjusting to it is costly and time-consuming. Therefore, many participants suggested that GuideGen should be implemented as a plug-in for the currently used tools, such as Jira or TFS, instead of being a standalone tool.

GuideGen assumes that traceability links between requirements and acceptance tests exist, which, to some extent, constitutes a limitation of our approach. However, none of the participants in our second evaluation mentioned this limitation as an obstacle to the usability of GuideGen.

We had limited the initial version of GuideGen to one acceptance test per requirement because we had found in a previous study about actual practices concerning requirements and acceptance tests [HBCG16] that there was just one acceptance test per requirement in most cases. In our evaluation, however, it turned out that the practitioners considered this limitation as a major shortcoming of GuideGen. They explained that sometimes, primarily due to lack of time or simplicity of a requirement, they indeed create only one acceptance test for a requirement. However, they would need to have the possibility to also have more than one acceptance test per requirement.

6.8.3 Lessons Learned

In this sub-section we present the main lessons learned from the two evaluations. Based on these lessons, we already have incorporated some changes in the current version of GuideGen, while others will be taken as a basis for our future work.

Changes of numerical values are relevant. One of the main take-home messages from the first evaluation is that changes of numerical values in requirements should be considered as relevant. We, therefore, extended our model and algorithms, so that numerical values are processed and, accordingly, suggestions are generated for this change type.

Some adaptations of user interface are needed for better usability. The user interface should be adapted so that test engineers can see suggestions directly in the form for editing an affected acceptance test. We have added this feature to GuideGen: in the current version, suggestions are shown next to the text of the affected test (see Figure 6.23). In our future work we will try to identify the exact steps within an acceptance test that are affected by changes in requirements.

The notification system should be adapted for more effective communication of changes. We improved our notification system based on this feedback. In the current version, GuideGen (1) generates e-mails when acceptance tests are updated (see Figure 6.24) or requirements are added (see Figure 6.15) and (2) shows warnings whenever documents are non-aligned due to a modification (see Figure 6.22) or an addition of a requirement (See Figure 6.14).

Solving the problem of overwhelming users with too many notification messages when requirements frequently change is subject to future work. We plan to make the notification system configurable so that, for example, users could opt for receiving only one or two messages per day which summarize all changes that have happened since the last notification.

A one-to-one relationship between requirements and their acceptance tests is not sufficient. From the second evaluation, we learned that supporting only one-to-one relationships between requirements and their acceptance tests is a major limitation. Therefore, we have extended the GuideGen tool so that it allows the creation of more than one acceptance test per requirement. However, the current version of GuideGen generates the same set of change suggestions for all acceptance tests when a requirement is changed. As a first step towards solving this problem, we have experimentally assessed whether existing change impact analysis tools can be used for identifying which change in a requirement affects which one of its associated acceptance tests (see Section 6.7). The results indicate that it is possible to extend GuideGen such that it generates individually tailored guidance for each of the affected tests. This is subject to future work.

GuideGen lacks many features. In the second evaluation we found that GuideGen is more applicable to smaller projects, as many features were missing. By adding the missing features we extended the applicability of GuideGen to projects with different types of documents, different roles with various responsibilities and different documentation management and tracking needs.

GuideGen would be more applicable as a plug-in than as a standalone tool. This is a major lesson we learned from the second evaluation. The main reasons are the missing features, the complex data migration process and an the reluctance of industrial practitioners to learn new tools. It is particularly relevant for big projects that already have a well established tool environment.

We will address this issue in our future work. For now, we improved GuideGen by adding the most relevant features that were missing, so that it can be used in projects for which data migration would not be too complex and in companies ready to adopt new tools.

Existing methods for change impact analysis have a potential to be used in GuideGen. The preliminary results from our experiment showed that the existing methods for change impact analysis among textual documents have a potential to be used for identifying affected acceptance tests based on changes in requirements. With careful choice of propagation conditions, NARCIA can be a good candidate for identifying the affected acceptance tests. Although further research on using ImpRec is needed, when choosing the best query parameters, ImpRec also showed good results and a potential to be used in GuideGen.

6.9 Related Work

Many researchers propose new ways to solve the problems of lacking traceability and inconsistencies between artifacts that occur when

systems evolve. For example, Antoniol et al. [ACC+02], Marcus and Maletic [MMS05], De Lucia et al. [LMOP12] and Hayes et al. [HDS06] use information retrieval methods to ensure automated traceability for change impact analyses.

Others employ natural language processing in order to identify artifacts impacted by changes in requirements. As previously explained, Arora et al. [ASG+15a] analyze the impact of changes in a requirement on other requirements in a system using NLP methods and implement a prototype tool, NARCIA. Borg et al. implemented ImpRec [Bor14], [BWRR17], a recommendation system for change impact analysis. The approach reuses the knowledge about previous traces to recommend development artifacts, other than source code, potentially impacted when resolving incoming issue reports.

All these approaches focus on identifying which requirements or other artifacts are impacted by a change in a requirement, while we investigate how to manage the change and which actions to perform in order to keep requirements and acceptance tests aligned. Yet these approaches can be used in GuideGen to identify the most affected acceptance test when many tests are related to a changed requirement.

Bridging the communication gap among people involved in developing a system draws attention of researchers and practitioners. Sinha et al. [SSC06] define and explain the communication problems when managing requirements in a distributed environment. Bjarnason and Sharp [BS17] and Adzic [Adz09] emphasize the communication problems between requirements engineers, developers and testers in agile

projects. By generating guidance in natural language that can be easily communicated to the interested parties via e-mail, our approach supports easy and timely communication of changes between requirements engineers and developers/testers.

In order to align requirements engineering and testing activities, researchers focused on defining formal models from informal requirements and automatically generating tests from these formal models [BEF11], [EGM+11], [PSM+09], [GR06], [GCFVP14]. For instance, Escalona et al. [EGM+11] provide an overview of the research with regard to generating tests from functional requirements. They conclude that in order to have a successful test generation, requirements would need to be specified in a much more formal way than they usually are in practice. Similarly, Post et al. [PSM+09] define a method for linking functional requirements to tests by creating a formal specification of requirements. Granda et al. provide an approach that automatically generates a set of abstract test cases from the requirements models.

Olsson and Grundy [MKS+16] implemented an approach that creates models from requirements descriptions, UML-style use case models and black-box test plans, which are later used for change impact analysis. However, their approach lacks automation and does not consider natural language, but formal models only.

Wang et. al automatically derive system test cases from Restricted Use Case models and domain models [WPG+15]. This work goes beyond other approaches for automated test generation from natural language requirements, such as [RPG03] or [KK06], as it generates tests that are

executable. However, it still relies on requirements documents that are structured in the form of use cases. For the companies that use use cases it would be interesting to combine this approach with GuideGen. In such a way, executable tests in natural language would be automatically derived from use cases and when the use cases change, GuideGen would automatically generate guidance on how to adapt the affected acceptance tests.

Since practitioners are often not keen on writing "code-like" or structured requirements, GuideGen aims at providing an approach that does not require practitioners to adapt the level of formalism in their requirements.

6.10 Conclusions

In this paper we provided a comprehensive overview of GuideGen, a tool-supported method for automatically generating guidance on how to align acceptance tests with evolving requirements. The evaluation results show that GuideGen generates suggestions with a correctness score of more than 80% for real-world agile requirements and around 67% for traditional requirements. This indicates that our approach provides useful guidance for maintaining acceptance tests and keeping them aligned with the evolving requirements. Practitioners perceive GuideGen as a practically useful tool, which can be applicable in practice, especially to smaller projects.

Based on the lessons learned in the two evaluations, we already have incorporated some changes in the current version of GuideGen. Other lessons provide directions for our future work.

Supporting one-to-many relationships between requirements and acceptance tests introduces the challenge of identifying which tests are actually affected by which changes in the requirement. The assessment of the two existing approaches for change impact analysis, NARCIA and ImpRec, showed that NARCIA could be used for change impact analysis with a correctness score of up to 90%, while the applicability of ImpRec would need to be investigated further.

Future Work. Based on the encouraging results of our experiment with existing change impact analysis tools, we plan to extend GuideGen such that it fully supports the case of requirements having more than one associated acceptance test. With this extension, GuideGen will be capable of generating individually tailored guidance for each of the affected acceptance tests. It might even be possible to highlight those parts within an affected acceptance test that need to be changed.

We also plan to make further adjustments to the notification system of GuideGen.

Further, we will explore how GuideGen can be implemented as a plug-in for existing commercial tools for managing requirements and acceptance tests. As a plug-in, GuideGen could be used easily within an existing tool ecosystem, without significant additional effort.

GuideGen only supports documents written in English. In the long run, we plan to explore how multiple languages could be supported with reasonable effort.

Chapter 7
Conclusion

Due to tight deadlines, poor communication of changes or inadequate tool support for an efficient change management, requirements and acceptance tests are often not kept aligned with each other nor with the actual software behavior.

In this thesis, we proposed a new approach for keeping requirements and acceptance test documents consistent and aligned. The core idea of our tool-based approach is to support change propagation between requirements and acceptance tests via automatically generated guidance. Guidance is a list of concrete suggestions about how to adapt acceptance tests when their corresponding requirements change. We derive such suggestions by analyzing the change patterns applied to a requirement and applying the appropriate rules for change propagation.

To prove the concept of our approach we implemented the GuideGen tool. Our tool analyzes change patterns applied to natural language

requirements written as free text without any prior formalization required. Based on the identified change patterns and defined rules for change propagation, GuideGen generates concrete suggestions about how to adapt the affected acceptance tests. The suggestions are intended to help test engineers to understand what has been changed in a requirement without an additional effort for browsing through different versions of the requirement or communicating with requirements engineers. Together with summarized changes applied to a requirement, the suggestions are communicated among all subscribed stakeholders (such as testers, product owners, project managers, etc.) via automatically generated e-mails and warnings, mitigating a manual effort or a need for additional meetings.

We conclude that with our approach, the process of propagating changes from requirements to acceptance tests is no longer a manual task, but rather a tool-supported, partially automated activity. Our approach goes beyond other approaches as it does not require an additional level of formalism when specifying requirements, but is, nevertheless, able to provide concrete suggestions on how to handle changes and what actions to perform in order to keep the affected acceptance tests consistent with their changed requirements.

According to Wieringa and Heerkens [WH06], the relevance of a solution may be judged based on its novelty, relevance for classes of world problems and relevance for theory. We consider our solution to be relevant, firstly, because it provides a novel method for a semi-automated change propagation between requirement and acceptance test documents. To our knowledge, our approach is unique in a sense that it provides concrete guidance in natural language that refer to actions in

acceptance tests and are based on changes applied to requirements written as free text. Secondly, our approach is real-world problem oriented. It contributes to solving a problem of maintaining software artifacts when systems evolve. Finally, our approach is relevant for theory, as it builds upon the existing research, but also provides new research directions in the field of requirements change management and change propagation.

7.1 Revisiting the Research Questions

The goal of this thesis is achieved by answering our research questions, presented in Section 1.2. We revisit each of the questions and summarize the answers to them, as follows.

RQ1. How are requirements and acceptance test documents currently maintained in practice?

In Chapter 2, we describe our exploratory study with fifteen practitioners from ten companies worldwide. The study assessed the current state of practice in terms of writing and maintaining requirements and acceptance test documents and was conducted in the form of semi-structured interviews.

The results show that the practice of maintaining documentation has evolved over time, especially with the spread of Agile development. In particular, practitioners rely mainly on verbal communication when

managing changes, although they are aware that such a way is error-prone and may lead to misunderstandings and, therefore, project delays and unintended costs. Further, we found that modern documentation management tools do not provide enough support for documentation update, especially in terms of documentation traceability and change propagation. Moreover, the practitioners emphasized a need for automation of these tasks.

Findings from this study, together with findings obtained from a review of related literature, provided a clear motivation for implementing a tool-based approach for supporting efficient communication and documentation change management. The approach is presented in Chapter 3.

RQ2. What is an effective tool-based approach for keeping requirements and acceptance tests consistent when system requirements evolve?

In order to support more efficient documentation management and facilitate communication of changes, in Chapter 3 we presented our novel approach for generating suggestions on how to adapt acceptance tests based on changes in their corresponding requirements. The approach relies on existing IR and NLP methods in order to successfully generate suggestions which are to a large extent correct, complete and clear. Moreover, the approach provides a notification system based on automatically generated e-mails and warnings that keeps all the stakeholders aware of the changes and mismatches between documents, ensuring that changes are communicated on time among different

stakeholders. By implementing such an approach, we contributed to mitigating the challenges found in the exploratory study, described in Chapter 2 and answered our second research question.

RQ3: How successful is our approach in generating natural language suggestions for adapting acceptance tests based on changes in requirements?

To answer this question, we evaluated the approach described in Chapter 3, by using real-world requirements and acceptance tests from three industrial projects. For 262 non-trivial changes patterns applied to the requirements, we generated suggestions about how to adapt acceptance tests and then seven experts evaluated the correctness, completeness, understandability and relevance of the suggestions. The results indicate that, with a correctness score of more than 80% for agile requirements and around 67% for traditional requirements, our approach provides useful guidance for keeping requirements and acceptance tests aligned when system requirements evolve.

With this evaluation we also revealed what improvements in our method are needed in order to achieve better results. In addition, by conducting a short qualitative assessment with the seven participants, we revealed what additional features are desired in order to improve the usability and increase the usefulness of GuideGen.

RQ4. How useful and applicable is our approach for real-world projects and organizations?

Based on the findings from the study described in Chapter 3, we improved our method and added more features to the tool. We then performed a more thorough qualitative evaluation of the approach and the tool in order to assess the general usefulness and applicability of GuideGen to real-world projects and organizations. The evaluation was conducted as semi-structured interviews with 23 participants from ten companies based in Europe. We describe this study in Chapter 5.

The results show that GuideGen is perceived as useful and applicable to smaller projects as it is, while further improvements are needed in order to make it applicable to more complex projects. In particular, some participants pointed out that GuideGen would be more useful if it were implemented as a plugin for the currently used documentation management tools, so that no data migration or additional effort for learning new tools is required in order to use the main features of GuideGen. Others stated that GuideGen can be used as a standalone tool, but it should then provide more features, such as a possibility to have one-to-many (and not only one-to-one) relationship between a requirement and its tests, to see the outcome of the test execution, to see the status of a requirement, etc.

Based on feedback from this study, we improved GuideGen by implementing the additional features, as described in Chapter 6. This raised some new challenges and research questions. For instance, by providing the possibility of relating more than one acceptance test to

a single requirement, a new challenge of identifying which of those tests are actually impacted by a change in the requirement is posed. As the existing research with respect to change impact analysis offers various solutions to this problem, we additionally derived a new research question that refers to the ability of the existing methods to identify affected acceptance tests based on changes in requirements, as follows.

RQ5. How well do the existing approaches for change impact analysis perform when applied between requirements and acceptance tests?

To answer this question, we conducted an experiment by using two of the existing approaches for change impact analysis between software artifacts written as free-text, NARCIA [ASG+15b] and ImpRec [BWRR17]. We applied these approaches on a small set of requirements and their multiple related acceptance tests. The dataset was obtained from an industrial, agile project and contains nine requirements and and twenty-three acceptance tests related to them. Our preliminary results show that NARCIA could be used for change impact analysis with a correctness score of up to 90%, while the applicability of ImpRec would need to be investigated further, as we were not able to use all its features due to small amount of input data in our dataset.

7.2 Next Steps

The first evaluation of GuideGen showed that the approach is able to generate suggestions correctly in about 80% for agile and 67% for traditional requirements. The suggestions were not correct due to different factors, such as the type of changes applied to a requirement or sentence complexity. For instance, when new sentences are added to a requirement, our approach identifies this change as relevant and generates suggestions accordingly, without previously checking whether the sentences indeed affect actions in acceptance tests. The added sentences may represent just additional notes or clarifications of terms that are introduced previously in the requirement. GuideGen currently does not detect nor filter out such sentences. Therefore, further research on more sophisticated identification of relevant and irrelevant changes is needed, especially in case when the change pattern *"addition of a whole sentence"* is applied to a requirement.

The complexity of changed sentences affects the quality of suggestions generated for the applied changes. For instance, when sentences contain one or more relative clauses or statements in parentheses, it might be challenging for GuideGen to correctly identify all the relevant words related to the changed words and to generate correct, complete and understandable suggestions. More thorough investigation of possible complex sentence structures is needed in order to provide a better correctness coverage for generated suggestions.

Our approach currently does not consider acceptance tests when generating suggestions, but it analyzes only changes applied to a requirement. Therefore, the generated suggestions are rather general. A

further research is needed in order to provide a more sophisticated method that would generate more concrete suggestions and point out the specific parts of the acceptance test that should be adapted according to changes applied to its related requirement.

Based on the results and feedback from our last evaluation of GuideGen, we adapted the approach and the tool, as described in Chapter 6. However, following the proposed, iterative research methodology, *the evaluation of the newly implemented features* is missing. As explained previously, adding the new features opened new challenges and new research directions. For instance, with our experiment, described in Chapter 6, we made a first step towards identifying acceptance tests affected by changes in their related requirements. However, further research is needed with regard to change impact analysis between requirements and their related acceptance tests. Moreover, our experiment included only a small dataset and further experiments with larger datasets are needed.

The results from our first evaluation showed to what extent GuideGen performs correctly when generating suggestions in natural language. With our second evaluation, we obtained opinions from practitioners about usefulness and applicability of GuideGen. In our future work it would be interesting to conduct a longitudinal study. With such a study we will further strengthen the validation of GuideGen by assessing the effectiveness, applicability and usefulness of the approach when applied in practice.

Finally, our approach is rule based. An interesting direction for future work is to assess to what extent machine learning approaches would

be able to generate correct suggestions for acceptance test adaptation based on changes in their associated requirements.

To summarize, our approach did not only contribute to solving the particular problem of keeping requirements and their tests consistent when system requirements evolve, but it also opened new interesting research directions in the field of change impact analysis, change propagation and natural language processing for requirements engineering.

Bibliography

[AAA+15] Khubaib Amjad Alam, Rodina Ahmad, Adnan Akhunzada, Mohd Hairul Nizam Md Nasir, and Samee U Khan. Impact analysis and change propagation in service-oriented enterprises: A systematic review. *Information Systems*, 54:43–73, 2015.

[AAW+16] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042*, 2016.

[ACC+02] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.

[Adz09] Gojko Adzic. *Bridging the communication gap: specification by example and agile acceptance testing*. Neuri, London, 2009.

[Adz16] Gojko Adzic. How to get the most out of Given-When-Then. `https://gojko.net/2015/02/25/how-to-get-the-most-out-of-given-when-then/`, 2016. [Online; accessed 08-Dec-2018].

[AFK16] Emil Alégroth, Robert Feldt, and Pirjo Kolström. Maintenance of automated test suites in industry: An empirical study on visual GUI testing. *Information and Software Technology*, 73:66–80, 2016.

[AHB16] Hussin Ahmed, Azham Hussain, and Fauziah Baharom. Current challenges of requirement change management. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 8(10):173–176, 2016.

[ASG+15a] Chetan Arora, Mehrdad Sabetzadeh, Arda Goknil, Lionel C Briand, and Frank Zimmer. Change impact analysis for natural language requirements: An NLP approach. In *23rd IEEE International Requirements Engineering Conference (RE'15)*, pages 6–15. IEEE, 2015.

[ASG+15b] Chetan Arora, Mehrdad Sabetzadeh, Arda Goknil, Lionel C Briand, and Frank Zimmer. NARCIA: an automated tool for change impact analysis in natural language requirements. In *10th European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 962–965. ACM, 2015.

[BA96] Shawn A Bohner and RS Arnold. *Software change impact analysis*. IEEE Computer Society Press, 1996.

[BB17] Elizabeth Bjarnason and Markus Borg. Aligning requirements and testing: Working together toward the same goal. *IEEE Software*, 34(1):20–23, 2017.

[BCKG15] Eya Ben Charrada, Anne Koziolek, and Martin Glinz. Supporting requirements update during software evolution. *Journal of Software: Evolution and Process*, 27(3):166–194, 2015.

[BEF11] Zeinab Alizadeh Barmi, Amir Hossein Ebrahimi, and Robert Feldt. Alignment of requirements specification and testing: A systematic mapping study. In *Fourth IEEE International Conference on Software Testing, Verification and Validation*, pages 476–485. IEEE, 2011.

[BGST12] Daniel Berry, Ricardo Gacitua, Pete Sawyer, and Sri Fatimah Tjong. The case for dumb requirements engineering tools. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 211–217. Springer, 2012.

[BGW13] Markus Borg, Orlena CZ Gotel, and Krzysztof Wnuk. Enabling traceability reuse for impact analyses: A feasibility study in a safety context. In *7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 72–78. IEEE, 2013.

[BMC+14] Sebastian Barney, Varun Mohankumar, Panagiota Chatzipetrou, Aybuke Aurum, Claes Wohlin, and Lefteris Angelis. Software quality across borders: Three case studies on company internal alignment. *Information and Software Technology*, 56(1):20–38, 2014.

[BMS02] Evelyn J Barry, Tridas Mukhopadhyay, and Sandra A Slaughter. Software project duration and effort: an empirical study. *Information Technology and Management*, 3(1-2):113–136, 2002.

[Boh96] Shawn A Bohner. Impact analysis in the software change process: a year 2000 perspective. In *International Conference on Software Maintenance*, pages 42–51. IEEE Computer Society, 1996.

[Bor14] Markus Borg. Embrace your issues: compassing the software engineering landscape using bug reports. In *29th ACM/IEEE International Conference on Automated Software Engineering*, pages 891–894. ACM, 2014.

[BRB+14] Elizabeth Bjarnason, Per Runeson, Markus Borg, Michael Unterkalmsteiner, Emelie Engström, Björn Regnell, Giedre Sabaliauskaite, Annabella Loconsole, Tony Gorschek, and Robert Feldt. Challenges and practices in aligning requirements with verification and validation: a case study of six companies. *Empirical Software Engineering*, 19(6):1809–1855, 2014.

[Bri03] Lionel C Briand. Software documentation: how much is enough? In *Seventh European Conference on Software Maintenance and Reengineering*, pages 13–15. IEEE, 2003.

[BS17] Elizabeth Bjarnason and Helen Sharp. The role of distances in requirements communication: a case study. *Requirements Engineering*, 22(1):1–26, 2017.

[BUEB15] Elizabeth Bjarnason, Michael Unterkalmsteiner, Emelie Engström, and Markus Borg. An industrial case study on test cases as requirements. In *International Conference on Agile Software Development*, pages 27–39. Springer, 2015.

[BWRR17] Markus Borg, Krzysztof Wnuk, Björn Regnell, and Per Runeson. Supporting change impact analysis using a recommendation system: An industrial case study in a safety-critical context. *IEEE Transactions on Software Engineering*, 43(7):675–700, 2017.

[CBC+15] Gustavo Carvalho, Flávia Barros, Ana Carvalho, Ana Cavalcanti, Alexandre Mota, and Augusto Sampaio. NAT2TEST tool: From natural language requirements to test cases based on CSP. In *Software Engineering and Formal Methods*, pages 283–290. Springer, 2015.

[CC17] John W Creswell and J David Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2017.

[CCTF17] Wanyu Chen, Honghui Chen, Chen Tao, and Cai Fei. FACIA: A Fully Automatic Change Impact Analysis Method for Large Scale Requirements. In *2017 International Conference on Applied Mechanics and Mechanical Automation (AMMA 2017)*, pages 419–427, 2017.

[CF96] J Chudge and D Fulton. Trust and co-operation in system development: applying responsibility modelling to the problem of changing requirements. *Software Engineering Journal*, 11(3):193–204, 1996.

[CH] Jan Schneider Chuck Hagenbuch. Text_diff-engine for performing and rendering text diffs. `https://pear.horde.org/`. [Online; accessed 20-June-2017].

[CHBC+07] Jane Cleland-Huang, Brian Berenbach, Stephen Clark, Raffaella Settimi, and Eli Romanova. Best practices for automated traceability. *IEEE Computer*, 40(6):27–35, 2007.

[CHCC03] Jane Cleland-Huang, Carl K Chang, and Mark Christensen. Event-based traceability for managing evolutionary change. *IEEE Transactions on Software Engineering*, 29(9):796–810, 2003.

[CK92] Michael G Christel and Kyo C Kang. Issues in requirements elicitation. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Institute, 1992.

[CKG12] Eya Ben Charrada, Anne Koziolek, and Martin Glinz. Identifying outdated requirements based on source code changes. In *2012 20th IEEE International Requirements Engineering Conference (RE)*, pages 61–70. IEEE, 2012.

[CKI88] Bill Curtis, Herb Krasner, and Neil Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1988.

[con] Confluence - a documentation management tool. `https://www.atlassian.com/software/confluence`. [Online; accessed 08-Dec-2018].

[CR08] Lan Cao and Balasubramaniam Ramesh. Agile requirements engineering practices: An empirical study. *IEEE Software*, 25(1):60–67, 2008.

[Dav89] Fred D Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 13(3):319–340, 1989.

[Dav93] Alan M Davis. *Software requirements: objects, functions, and states*. Prentice-Hall, 1993.

[DC06] Daniela Damian and James Chisan. An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. *IEEE Transactions on Software Engineering*, 32(7):433–453, 2006.

[dCdMBH15] Marcelo Moreira Holanda de Castro, Juliana de Melo Bezerra, and Celso Massaki Hirata. A CNL for requirements as the basis to automate tasks of critical system development. In *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, pages 82–89. IEEE, 2015.

[DEW+18] Flemström Daniel, Enoiu Eduard, Azal Wasif, Sundmark Daniel, Gustafsson Thomas, and Kobetski Avenir. From natural language requirements to passive test cases using guarded assertions. In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 470–481. IEEE, 2018.

[DFG07] Nina Dzamashvili-Fogelström and Tony Gorschek. Test-case driven versus checklist-based inspections of software requirements–an experimental evaluation. In *10th Workshop on Requirements Engineering (WER 07)*, 2007.

[DLFOT04] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Enhancing an artefact management system with traceability recovery features. In *20th International Conference on Software Maintenance, 2004.*, pages 306–315. IEEE, 2004.

[DM06] Siqing Du and Douglas P Metzler. An automated multi-component approach to extracting entity relationships from database requirement specification documents. In *International Conference on Application of Natural Language to Information Systems*, pages 1–11. Springer, 2006.

[dSJV12] Valdivino Alexandre de Santiago Junior and Nandamudi Lankalapalli Vijaykumar. Generating model-based test cases from natural language requirements for space application software. *Software Quality Journal*, 20(1):77–143, 2012.

[DWP15] Marian Daun, Thorsten Weyer, and Klaus Pohl. Detecting and correcting outdated requirements in function-centered engineering of embedded systems. In *21st International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2015)*, pages 65–80. Springer, 2015.

[DWW+14] Bogdan Dit, Michael Wagner, Shasha Wen, Weilin Wang, Mario Linares-Vásquez, Denys Poshyvanyk, and Huzefa Kagdi. Impactminer: A tool for change impact analysis. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 540–543, New York, NY, USA, 2014. ACM.

[EGM+11] María José Escalona, Javier Jesús Gutierrez, Manuel Mejías, Gustavo Aragón, Isabel Ramos, Jesús Torres, and Francisco José Domínguez. An overview on test generation from functional requirements. *Journal of Systems and Software*, 84(8):1379–1393, 2011.

[ESSD08] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer, 2008.

[ESSD13] Hamzeh Eyal-Salman, Abdelhak-Djamel Seriai, and Christophe Dony. Feature-to-code traceability in a collection of software variants: Combining formal concept analysis and information retrieval. In *14th International Conference on Information Reuse and Integration (IRI 2013)*, pages 209–216. IEEE, 2013.

[GB94] Leah Goldin and Daniel M Berry. Abstfinder, a prototype abstraction finder for natural language text for use in requirements elicitation: design, methodology, and evaluation. In *Proceedings of IEEE International Conference on Requirements Engineering*, pages 84–93. IEEE, 1994.

[GCFVP14] Maria Fernanda Granda, Nelly Condori-Fernández, Tanja EJ Vos, and Oscar Pastor. Towards the automated generation of abstract test cases from requirements models. In *1st International Workshop on Requirements Engineering and Testing (RET), 2014*, pages 39–46. IEEE, 2014.

[GR06] Ankit Goel and Abhik Roychoudhury. Synthesis and traceability of scenario-based executable models. In *Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2006)*, pages 347–354. IEEE, 2006.

[Gra02] Dorothy Graham. Requirements and testing: Seven missing-link myths. *IEEE Software*, 19(5):15–17, 2002.

[GSG10] Ricardo Gacitua, Pete Sawyer, and Vincenzo Gervasi. On the effectiveness of abstraction identification in requirements engineering. In *18th IEEE International Requirements Engineering Conference*, pages 5–14. IEEE, 2010.

[GW89] Donald C Gause and Gerald M Weinberg. *Exploring requirements: quality before design*. Dorset House, New York, 1989.

[Hal03] Marty Hall. *Core servelet and JavaServer Pages, online version*. Sun Microsystems Press available at: http://www. coreservlets. com, 2003.

[Han97] Jun Han. Supporting impact analysis and change propagation in software engineering environments. In *Software Technology and Engineering Practice, 1997. Proceedings., Eighth IEEE International Workshop on [incorporating Computer Aided Software Engineering]*, pages 172–182. IEEE, 1997.

[HBCG16] Sofija Hotomski, Eya Ben Charrada, and Martin Glinz. An exploratory study on handling requirements and acceptance test documentation in industry. In *24th IEEE International Requirements Engineering Conference (RE'16)*, pages 116–129. IEEE, 2016.

[HBCG17] Sofija Hotomski, Eya Ben Charrada, and Martin Glinz. Aligning requirements and acceptance tests via automatically generated guidance. In *4th Workshop on Requirements Engineering and Testing (RET)*, 2017.

[HBCG18] Sofija Hotomski, Eya Ben Charrada, and Martin Glinz. Keeping evolving requirements and acceptance tests aligned with automatically generated guidance. In *24th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'18)*, pages 247–264. Springer, 2018.

[HDS06] Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Transactions on Software Engineering*, 32(1):4–19, 2006.

[HDS+07] Jane Huffman Hayes, Alex Dekhtyar, Senthil Karthikeyan Sundaram, E Ashlee Holbrook, Sravanthi Vadlamudi, and Alain April. Requirements tracing on target (retro): improving software maintenance through traceability recovery. *Innovations in Systems and Software Engineering*, 3(3):193–202, 2007.

[Heu01] Jim Heumann. Generating test cases from use cases. *The Rational Edge*, 6(01), 2001.

[HF10] Jez Humble and David Farley. *Continuous delivery: Reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[HG03] H.M. Harmain and Robert Gaizauskas. CM-builder: A natural language-based case tool for object-oriented analysis. *Automated Software Engineering*, 10(2):157–181, 2003.

[HG18a] Sofija Hotomski and Martin Glinz. GuideGen: a tool for keeping requirements and acceptance tests aligned. In *Proceedings of the 40th International Conference on Software Engineering (ICSE 2018), Companion volume*, pages 49–52. ACM, 2018.

[HG18b] Sofija Hotomski and Martin Glinz. A qualitative study on using GuideGen to keep requirements and acceptance tests aligned. In *26th IEEE International Requirements Engineering Conference (RE'18)*, pages 29–39. IEEE, 2018.

[HG19] Sofija Hotomski and Martin Glinz. Guidegen: An approach for keeping requirements and acceptance tests aligned via automatically generated guidance. *Information and Software Technology*, 110:17–38, 2019.

[HGB08] Bill Hasling, Helmut Goetz, and Klaus Beetz. Model based testing of system requirements using UML use case models. In *1st international conference on Software Testing, Verification, and Validation, 2008*, pages 367–376. IEEE, 2008.

[HH08] Børge Haugset and Geir Kjetil Hanssen. Automated acceptance testing: A literature review and an industrial case study. In *AGILE'08. Conference*, pages 27–38. IEEE, 2008.

[HKS97] Pei Hsia, David Kung, and Chris Sell. Software requirements and acceptance testing. *Annals of Software Engineering*, 3(1):291–317, 1997.

[HS12] Børge Haugset and Tor Stalhane. Automated acceptance testing as an agile requirements engineering practice. In *45th Hawaii International Conference on System Sciences*, pages 5289–5298. IEEE, 2012.

[IA10] Mohd Ibrahim and Rodina Ahmad. Class diagram extraction from textual requirements using natural language processing (NLP) techniques. In *2nd International Conference onComputer Research and Development, 2010*, pages 200–204. IEEE, 2010.

[IK11] Eeva-Mari Ihantola and Lili-Anne Kihn. Threats to validity and reliability in mixed methods accounting research. *Qualitative Research in Accounting & Management*, 8(1):39–58, 2011.

[IML09] Juha Itkonen, Mika V Mantyla, and Casper Lassenius. How do testers do it? an exploratory study on manual testing practices. In *3rd International Symposium on Empirical Software Engineering and Measurement*, pages 494–497. IEEE, 2009.

[jir] Plan, track, support with Jira. `https://www.atlassian.com/software/jira`. [Online; accessed 08-Dec-2018].

[jsp] Java server pages. `http://www.oracle.com/technetwork/java/index-jsp-138231.html`. [Online; accessed 12-July-2018].

[KHO17] Pertti Karhapää, Alireza Haghighatkhah, and Markku Oivo. What do we know about alignment of requirements engineering and software testing? In *21st International Conference on Evaluation and Assessment in Software Engineering*, pages 354–363. ACM, 2017.

[KIK+10] Henri Karhatsu, Marko Ikonen, Petri Kettunen, Fabian Fagerholm, and Pekka Abrahamsson. Building blocks for self-organizing software development teams a framework model and empirical pilot study. In *2nd International Conference on Software Technology and Engineering (ICSTE), 2010*, volume 1, pages V1–297. IEEE, 2010.

[Kit96] Barbara Ann Kitchenham. Evaluating software engineering methods and tool. part 1: The evaluation context and evaluation methods. *ACM SIGSOFT Software Engineering Notes*, 21(1):11–15, 1996.

[KK06] Mika Katara and Antti Kervinen. Making model-based testing more agile: a use case driven approach. In *Haifa Verification Conference*, pages 219–234. Springer, 2006.

[KVKU09] Johanna Kukkanen, Kyösti Väkeväinen, Marjo Kauppinen, and Eero Uusitalo. Applying a systematic approach to link requirements and testing: A case study. In *Proceedings of the 16th Asia-Pacific Software Engineering Conference*, pages 482–488. IEEE Computer Society, 2009.

[LB14] Jacob Larsson and Markus Borg. Revisiting the challenges in aligning re and v&v: Experiences from the public sector. In *1st International Workshop on Requirements Engineering and Testing (RET)*, pages 4–11. IEEE, 2014.

[LDvdWB16] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn EM van der Werf, and Sjaak Brinkkemper. Improving agile requirements: the quality user story framework and tool. *Requirements Engineering*, 21(3):383–403, 2016.

[Lef10] Dean Leffingwell. *Agile software requirements: lean requirements practices for teams, programs, and the enterprise*. Addison-Wesley Professional, 2010.

[Leh11] Steffen Lehnert. *A Review of Software Change Impact Analysis*. Technical Report, Universitaetsbibliothek Ilmenau, 2011.

[LFOT07] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 16(4):13, 2007.

[LMOP12] Andrea Lucia, Andrian Marcus, Rocco Oliveto, and Denys Poshyvanyk. Information retrieval methods for automated traceability recovery. *Software and Systems Traceability*, pages 71–98, 2012.

[LMP04] Mich Luisa, Franch Mariangela, and Novi Inverardi Pierluigi. Market research for requirements analysis using linguistic tools. *Requirements Engineering*, 9(1):40–56, 2004.

[LRD+17] Garm Lucassen, Marcel Robeer, Fabiano Dalpiaz, Jan Martijn E. M. van der Werf, and Sjaak Brinkkemper. Extracting conceptual models from user stories with visual narrator. *Requirements Engineering*, 22(3):339–358, Sep 2017.

[LSF03] Timothy C Lethbridge, Janice Singer, and Andrew Forward. How software engineers use documentation: The state of the practice. *IEEE Software*, 20(6):35–39, 2003.

[LVD06] Marco Lormans and Arie Van Deursen. Can LSI help reconstructing requirements traceability in design and test? In *10th European Conference on Software Maintenance and Reengineering (CSMR), 2006.*, pages 10–pp. IEEE, 2006.

[MGND17] Kaushik Madala, Danielle Gaither, Rodney Nielsen, and Hyunsook Do. Automated identification of component state transition model elements from requirements. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 386–392. IEEE, 2017.

[MKS+16] Nor Aiza Moketar, Massila Kamalrudin, Safiah Sidek, Mark Robinson, and John Grundy. An automated collaborative requirements engineering tool for better validation of requirements. In *31st IEEE/ACM International Conference on Automated Software Engineering (ASE), 2016*, pages 864–869. IEEE, 2016.

[MM04] Grigori Melnik and Frank Maurer. Direct verbal communication as a catalyst of agile knowledge sharing. In *Agile Development Conference*, pages 21–31, 2004.

[MM07] Grigori Melnik and Frank Maurer. Multiple perspectives on executable acceptance test-driven development. In *International Conference on Extreme Programming and Agile Processes in Software Engineering*, pages 245–249. Springer, 2007.

[MM08] Robert C Martin and Grigori Melnik. Tests and requirements, requirements and tests: A möbius strip. *IEEE Software*, 25(1), 2008.

[MMC06] Grigori Melnik, Frank Maurer, and Mike Chiasson. Executable acceptance tests for communicating business requirements: customer perspective. In *Agile Conference, 2006*, pages 12–pp. IEEE, 2006.

[MMS05] Andrian Marcus, Jonathan I Maletic, and Andrey Sergeyev. Recovery of traceability links between software documentation and source code. *International Journal of Software Engineering and Knowledge Engineering*, 15(05):811–836, 2005.

[MNA05] Lars Mathiassen, Ojelanki K Ngwenyama, and Ivan Aaen. Managing change in software process improvement. *IEEE Software*, 22(6):84–91, 2005.

[MP12] Priyanka More and Rashmi Phalnikar. Generating uml diagrams from natural language specifications. *International Journal of Applied Information Systems*, 1(8):19–23, 2012.

[MRM04] Grigori Melnik, Kris Read, and Frank Maurer. Suitability of fit user acceptance tests for specifying functional requirements: Developer perspective. In *Conference on Extreme Programming and Agile Methods*, pages 60–72. Springer, 2004.

[MSB11] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 3rd edition edition, 2011.

[MSB+14] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The Stanford coreNLP natural language processing toolkit. In *ACL (System Demonstrations)*, pages 55–60, 2014.

[NBR11] John Noll, Sarah Beecham, and Ita Richardson. Global software development and collaboration: Barriers and solutions. *ACM Inroads*, 1(3):66–78, 2011.

[NdlVS13] Sunil Nair, Jose Luis de la Vara, and Sagar Sen. A review of traceability research at the requirements engineering conference RE@ 21. In *21st IEEE International Requirements Engineering Conference (RE'13)*, pages 222–229. IEEE, 2013.

[NZF04] Nur Nurmuliani, Didar Zowghi, and Sue Fowell. Analysis of requirements volatility during software development life cycle. *IEEE Australian Software Engineering Conference*, pages 1–10, 2004.

[Pat90] Michael Quinn Patton. *Qualitative evaluation and research methods*. SAGE Publications, 1990.

[PSM+09] Hendrik Post, Carsten Sinz, Florian Merz, Thomas Gorges, and Thomas Kropf. Linking functional requirements and software verification. In *17th IEEE International Requirements Engineering Conference (RE'09)*, pages 295–302. IEEE, 2009.

[Pug10] Ken Pugh. *Lean-Agile acceptance test-driven-development*. Pearson Education, 2010.

[qac] QAComplete for efficient test management. `https://qacomplete.com/`. [Online; accessed 08-Dec-2018].

[RFG+17] Benedetta Rosadini, Alessio Ferrari, Gloria Gori, Alessandro Fantechi, Stefania Gnesi, Iacopo Trotta, and Stefano Bacherini. Using NLP to detect requirements defects:

an industrial experience in the railway domain. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 344–360. Springer, 2017.

[RG+00] Johannes Ryser, Martin Glinz, et al. Scent: A method employing scenarios to systematically derive test cases for system test. *University of Zurich*, 2000.

[RH08] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131, Dec 2008.

[RLB+13] Vasile Rus, Mihai C Lintean, Rajendra Banjade, Nobal B Niraula, and Dan Stefanescu. Semilar: The semantic similarity toolkit. In *ACL (Conference System Demonstrations)*, pages 163–168, 2013.

[RPG03] Matthias Riebisch, Ilka Philippow, and Marco Götze. UML-based statistical test case generation. *Objects, Components, Architectures, Services, and Applications for a Networked World*, pages 394–411, 2003.

[RST02] Bob Ryan, Robert William Scapens, and Michael Theobald. *Research Method and Methodology in Finance and Accounting*. London: Thomson, 2002.

[RTCT07] Filippo Ricca, Marco Torchiano, Mariano Ceccato, and Paolo Tonella. Talking tests: an empirical assessment of

the role of fit acceptance tests in clarifying requirements. In *9th International Workshop on Principles of Software Evolution (IWPSE 2007)*, pages 51–58. ACM, 2007.

[RTD+09] Filippo Ricca, Marco Torchiano, Massimiliano Di Penta, Mariano Ceccato, and Paolo Tonella. Using acceptance tests as a support for clarifying requirements: A series of experiments. *Information and Software Technology*, 51(2):270–283, 2009.

[Rya93] Kevin Ryan. The role of natural language in requirements engineering. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 240–242. IEEE, 1993.

[SBG17] Najia Saher, Fauziah Baharom, and Osman Ghazali. Requirement change taxonomy and categorization in agile software development. In *6th International Conference on Electrical Engineering and Informatics (ICEEI), 2017*, pages 1–6. IEEE, 2017.

[ser] Java servlet technology. `http://www.oracle.com/technetwork/java/index-jsp-135475.html`. [Online; accessed 05-July-2018].

[SGB14] Richa Sharma, Sarita Gulia, and KK Biswas. Automated generation of activity and sequence diagrams from natural language requirements. In *International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), 2014*, pages 1–9. IEEE, 2014.

[SH11] Christoph Johann Stettina and Werner Heijstek. Necessary and neglected? An empirical study of internal documentation in agile software development teams. *29th ACM International Conference on Design of Communication*, pages 159–166, 2011.

[Sin98] Janice Singer. Practices of software maintenance. *Proceedings International Conference on Software Maintenance*, pages 1–7, 1998.

[SLE+10] Giedre Sabaliauskaite, Annabella Loconsole, Emelie Engström, Michael Unterkalmsteiner, Björn Regnell, Per Runeson, Tony Gorschek, and Robert Feldt. Challenges in aligning requirements engineering and verification in a large-scale industrial context. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 128–142. Springer, 2010.

[SOSA99] George E Stark, Paul Oman, Alan Skillicorn, and Alan Ameele. An examination of the effects of requirements changes on software maintenance releases. *Journal of Software Maintenance: Research and Practice*, 11(5):293–309, 1999.

[SS97] Ian Sommerville and Pete Sawyer. *Requirements engineering: a good practice guide*. John Wiley&Sons, 1997.

[SSC06] Vibha Sinha, Bikram Sengupta, and Satish Chandra. Enabling collaboration in distributed requirements management. *IEEE Software*, 23(5):52–61, 2006.

[tes] TestLink - a tool for test management. `http://testlink.org/`. [Online; accessed 08-Dec-2018].

[tfs] Team Foundation Server (TFS) Administration. `https://docs.microsoft.com/en-us/tfs/server/`. [Online; accessed 08-Dec-2018].

[UFG14] Michael Unterkalmsteiner, Robert Feldt, and Tony Gorschek. A taxonomy for requirements engineering and software test alignment. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(2):16, 2014.

[UKKD08] Eero J Uusitalo, Marko Komssi, Marjo Kauppinen, and Alan M Davis. Linking requirements and testing in practice. In *16th IEEE International Requirements Engineering Conference (RE'16)*, pages 265–270. IEEE, 2008.

[WB13] Karl Wiegers and Joy Beatty. *Software requirements*. Pearson Education, 2013.

[WGC+04] Chanoch Wiggers, Ben Galbraith, Vivek Chopra, Sing Li, Debashish Bhattacharjee, Amit Bakore, Romin Irani, Sandip Bhattacharya, and Chad Fowler. *Professional Apache Tomcat*. John Wiley & Sons, 2004.

[WH94] Jim Welsh and Jun Han. Software documents: Concepts and tools. *Software - Concepts and Tools*, 15(1):12–25, 1994.

[WH06] Roel J Wieringa and Johannes MG Heerkens. The methodological soundness of requirements engineering papers: a conceptual framework and two case studies. *Requirements Engineering*, 11(4):295–307, 2006.

[WHT17] Matt Wynne, Aslak Hellesoy, and Steve Tooke. *The cucumber book: behaviour-driven development for testers and developers*. Pragmatic Bookshelf, 2017.

[WPG+15] Chunhui Wang, Fabrizio Pastore, Arda Goknil, Lionel Briand, and Zohaib Iqbal. Automatic generation of system test cases from use case specifications. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, pages 385–396. ACM, 2015.

[WRH+12] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[YBL13] Tao Yue, Lionel C Briand, and Yvan Labiche. Facilitating the transition from use case models to analysis models: Approach and experiments. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 22(1):5, 2013.

[YSJ10] Koen Yskout, Riccardo Scandariato, and Wouter Joosen. Change patterns: Co-evolving requirements and architecture. Technical report, Department of Computer Science, Leuven, Belgium, 2010.

Appendix A
Publications

This appendix presents the list of publications on which this cumulative dissertation is built. A publication that is not included in the dissertation is marked as such.

A.1 Conference Papers

[HBCG18b] Sofija Hotomski, Eya Ben Charrada, and Martin Glinz. An exploratory study on handling requirements and acceptance test documentation in industry. In 24th IEEE International Requirements Engineering Conference (RE'16), pages 116–129. IEEE, 2016. DOI: 10.1109/RE.2016.37

[HBCG17] Sofija Hotomski, Eya Ben Charrada, and Martin Glinz. Aligning requirements and acceptance tests via auto-matically generated guidance. In 4th Workshop on Requirements Engineering

and Testing (RET), pages 339–342, 2017. DOI: 10.1109/REW.2017.37
(This publication is a precursor of [HBCG18], reporting preliminary results. Therefore, it was not included in this thesis.)

[HBCG18] Sofija Hotomski, Eya Ben Charrada, and Martin Glinz. Keeping evolving requirements and acceptance tests aligned with automatically generated guidance. In 24th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'18), pages 247–264. Springer, 2018. DOI: 10.1007/978-3-319-77243-1_15

[HG18a] Sofija Hotomski and Martin Glinz. GuideGen: a tool for keeping requirements and acceptance tests aligned. In Proceedings of the 40th International Conference on Software Engineering (ICSE 2018), Companion volume, pages 49–52. ACM, 2018. DOI: 10.1145/3183440.3183484

[HG18b] Sofija Hotomski and Martin Glinz. A qualitative study on using GuideGen to keep requirements and acceptance tests aligned. In 26th IEEE International Requirements Engineering Conference (RE'18), pages 29–39, IEEE, 2018. DOI: 10.1109/RE.2018.00-54

A.2 Journal article

[SG18] Sofija Hotomski and Martin Glinz. GuideGen: An Approach for Keeping Requirements and Acceptance Tests Aligned via Automatically Generated Guidance. Information and Software Technology (IST) 110: 17–38, 2019. DOI: 10.1016/j.infsof.2019.01.011

Curriculum Vitae

Sofija Hotomski

Date of Birth:	September 21, 1987

Education

April 15 - April 19	PhD program in Computer Science, University of Zurich, Switzerland
October 10 - December 11	Master in Electrical and Computer Engineering, Faculty of Technical Sciences, Novi Sad, Serbia
October 06 - September 10	Bachelor of Science in Electrical and Computer Engineering, Faculty of Technical Sciences, Novi Sad, Serbia

Professional Experience

April 15 - April 19	Researcher and teaching assistant, University of Zurich, Switzerland
February 12 - March 15	SCADA Lead at Schneider Electric DMS NS, Novi Sad, Serbia
October 09 - February 12	Software Engineer at DMS Group, Novi Sad, Serbia